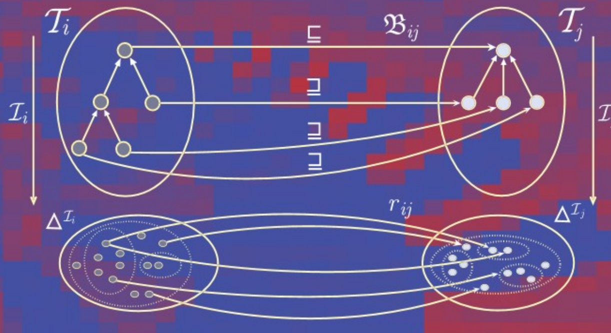
State-of-the-Art Survey

LNCS 5445

Heiner Stuckenschmidt
Christine Parent
Stefano Spaccapietra (Eds.)

Modular Ontologies

Concepts, Theories and Techniques for Knowledge Modularization



Springer

Heiner Stuckenschmidt Christine Parent
Stefano Spaccapietra (Eds.)

Modular Ontologies

Concepts, Theories and Techniques
for Knowledge Modularization

Springer

Volume Editors

Heiner Stuckenschmidt
Universität Mannheim, Institut für Informatik
B6, 26, 68159 Mannheim, Germany
E-mail: heiner@informatik.uni-mannheim.de

Christine Parent
Université de Lausanne, HEC ISI
1015 Lausanne, Switzerland
E-mail: christine.parent@epfll.ch

Stefano Spaccapietra
École Polytechnique Fédérale de Lausanne, EPFL-IC, Database Laboratory
1015 Lausanne, Switzerland
E-mail: stefano.spaccapietra@epfl.ch

Library of Congress Control Number: Applied for

CR Subject Classification (1998): D.1, D.2, D.3, F.3.2, F.4.1

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

ISSN	0302-9743
ISBN-10	3-642-01906-4 Springer Berlin Heidelberg New York
ISBN-13	978-3-642-01906-7 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12666959 06/3180 5 4 3 2 1 0

Preface

The growing emphasis on complexity concerns for ontologies has attracted significant interest from both the researchers' and the practitioners' communities in modularization techniques as a way to decrease the complexity of managing huge ontologies. Research has produced many complementary and competing approaches, mainly with the goal of supporting practitioners' methodologies with sound and precisely defined foundations and alternatives. Existing prototypes substantiate research results and experimental evaluations have been performed. Thus, a large body of work is available.

This book has been designed to provide the reader with a detailed analysis of where we stand today and which concepts, theories and techniques for knowledge modularization we can confidently rely on. The material for the book has been selected from research achievements that are mature enough to be considered as a firm and reliable basis on which to inspire further work and to develop solutions in concrete environments.

The content of the book has been organized in three parts. Part I holds a general introduction to the idea and issues characterizing modularization. This is followed by three chapters that offer an in-depth analysis of properties, criteria and knowledge import techniques for modularization. The last chapter discusses one of the three approaches that implement the modularization idea. The two other approaches are covered in detail in parts II and III.

Part II describes four major research proposals for creating modules from an existing ontology, either by partitioning an ontology into a collection of modules or by extracting one or more modules from the ontology. In both cases the knowledge in a module is a subset of the knowledge in the ontology.

Part III reports on collaborative approaches where modules that pre-exist (as independent ontologies) are linked together through mappings to form a virtual large ontology, called a distributed ontology. The first chapter discusses one of the core issues, the various kinds of languages for defining the mappings between elements of the modules. The following chapters of Part III describes three major alternative techniques for interconnecting ontologies in view of providing enriched knowledge to users of the collaborative system. (Please refer to the Introduction to Part III for more details.)

Parts II and III deal with ontologies only. Similar work in the database domain is well known and well documented in available text books. The material in this book was carefully reviewed before publication. We hope it will prove to be very helpful to anybody interested in knowledge modularization.

October 2008

Heiner Stuckenschmidt
Christine Parent
Stefano Spaccapietra

Contents

Part I Modularization Approaches

Introduction to Part I . . . 3

1 An Overview of Modularity
Christine Parent, Stefano Spaccapietra . . . 5

2 Formal Properties of Modularisation
Boris Konev, Carsten Lutz, Dirk Walther, Frank Wolter . . . 25

3 Criteria and Evaluation for Ontology Modularization Techniques
Mathieu d'Aquin, Anne Schlicht, Heiner Stuckenschmidt, Marta Sabou . . . 67

4 On Importing Knowledge from Ontologies
Alexander Borgida . . . 91

5 Modularity in Databases
Christine Parent, Stefano Spaccapietra, Esteban Zimányi . . . 113

Part II Partitioning and Extraction of Modules

Introduction to Part II . . . 157

6 Extracting Modules from Ontologies: A Logic-Based Approach
Bernardo Cuenca Grau, Ian Horrocks, Yevgeny Kazakov, Ulrike Sattler . . . 159

7 Structure-Based Partitioning of Large Ontologies
Heiner Stuckenschmidt, Anne Schlicht 187

8 Web Ontology Segmentation: Extraction, Transformation, Evaluation
Julian Seidenberg 211

9 Traversing Ontologies to Extract Views
Natalya F. Noy, Mark A. Musen 245

Part III Connecting Existing Ontologies

Introduction to Part III 263

10 Formal and Conceptual Comparison of Ontology Mapping Languages
Saartje Brockmans, Peter Haase, Luciano Serafini, Heiner Stuckenschmidt 267

11 Ontology Integration Using $\mathcal{E}$-Connections
Bernardo Cuenca Grau, Bijan Parsia, Evren Sirin 293

12 Composing Modular Ontologies with Distributed Description Logics
Luciano Serafini, Andrei Tamilin 321

13 Package-Based Description Logics
Jie Bao, George Voutsadakis, Giora Slutzki, Vasant Honavar 349

Author Index 373

Subject Index 375

List of Contributors

Jie Bao
Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12180, USA

Alexander Borgida
Dept. of Computer Science, Rutgers University, USA

Saartje Brockmans
Institute AIFB, University of Karlsruhe D-76128 Karlsruhe, Germany

Bernardo Cuenca Grau
University of Oxford, UK

Mathieu d’Aquin
Knowledge Media Institute (KMi) The Open University, Milton Keynes, United Kingdom

Peter Haase
Institute AIFB, University of Karlsruhe D-76128 Karlsruhe, Germany

Vasant Honavar
Department of Computer Science, Iowa State University, Ames, IA 50011, USA

Ian Horrocks
University of Oxford, UK

Yevgeny Kazakov
University of Manchester, UK

Boris Konev
University of Liverpool, UK

Carsten Lutz
University of Liverpool, UK

Mark A. Musen
Stanford Center for Biomedical Informatics Research, Stanford University

Natalya F. Noy
Stanford Center for Biomedical Informatics Research, Stanford University

Christine Parent
HEC ISI, Université de Lausanne, CH-1015 Lausanne, Switzerland

Bijan Parsia
University of Oxford, UK

Marta Sabou
Knowledge Media Institute (KMi) The Open University, Milton Keynes, United Kingdom

Ulrike Sattler
University of Manchester, UK

Anne Schlicht
Universität Mannheim, Germany

Julian Seidenberg
Bio-Health Informatics Group University of Manchester, United Kingdom

Luciano Serafini
Data and Knowledge Management Unit, Center for Information Technology - Irst, Fondazione Bruno Kessler, Via Sommarive 18, 38100 Povo di Trento, Italy

Evren Sirin
University of Oxford, UK

Giora Slutzki
Department of Computer Science, Iowa State University, Ames, IA 50011, USA

Stefano Spaccapietra
Database Laboratory, Ecole Polytechnique Fédérale de Lausanne, EPFL-IC, CH-1015

Heiner Stuckenschmidt
Universität Mannheim, Germany

Andrei Tamilin
Data and Knowledge Management Unit, Center for Information Technology - Irst, Fondazione Bruno Kessler, Via Sommarive 18, 38100 Povo di Trento, Italy

George Voutsadakis
Department of Computer Science, Iowa State University, Ames, IA 50011, USA

Dirk Walther
University of Liverpool, UK

Frank Wolter
University of Liverpool, UK

Esteban Zimányi
Université Libre de Bruxelles, Belgium

Part I

Modularization Approaches

Introduction to Part I

This part is meant to convey a general introduction to the domain of knowledge modularization. Chapter 1 overviews the issues and the proposed solutions that are relevant to ontology modularization. This chapter is deliberately informal to be accessible to the largest audience, so that readers can quickly get familiar with what will make up the rest of the book. Questions addressed in this initial chapter range from what is a module and how it can be characterized and assessed, to an overview of strategies to build modules and link them whenever needed. Chapters 2 to 4 explore in deeper detail and more formally specific issues that are nevertheless generic in the sense that they can be discussed independently from a specific approach to modularization. Chapter 2 establishes a formal characterization of the concept of module, introducing formal concepts that allow making a clear distinction among different types of modules and different relationships between the modules and the ontologies they come from. Each type of module exhibits different properties in terms of its potential functionalities, so it is important for a module designer to understand exactly how to proceed to get the desired modularity framework. Chapter 3 reports on an experimentation with various modularity techniques in various use case scenarios. Results show significant differences, namely in the size of the modules produced by the modularization techniques. They also provide a feedback on the qualities of the techniques. Next, Chapter 4 investigates how knowledge can be imported into a module from an external source. The aim of the chapter is to establish a clear and sound classification of various import techniques, which again should support designers with ways to choose the most appropriate technique given the goals to be achieved. Finally, Chapter 5 contains the detailed description of the Mads database modularization approach that currently best represents the efforts from the database community in terms of modularization techniques. It also includes, for comparison purposes, a short overview of one of the oldest approaches to modular ontologies, namely the Cyc project. This project represents the main achievement in terms of approaches where the definition and building of the modules and of the ontology they belong to are done in parallel at the same time. A short comparison between Cyc and Mads is also provided.

October 2008

Heiner Stuckenschmidt
Christine Parent
Stefano Spaccapietra

1

An Overview of Modularity

Christine Parent[1] and Stefano Spaccapietra[2]

[1] HEC ISI, Université de Lausanne, Switzerland
`christine.parent@epfl.ch`
[2] Database Laboratory, Ecole Polytechnique Fédérale de Lausanne, Switzerland
`stefano.spaccapietra@epfl.ch`

Summary. Modularization is a familiar concept in IT, widely used in e.g. software development. It has been somehow neglected in knowledge management. Database research has only marginally addressed the topic, mainly for the development of cooperative database systems. Instead, research on ontologies, seen as the knowledge provider for the semantic web, has significantly invested on modularization techniques, seen as a key capability in the current efforts towards scalable solutions that will enable ontologies to grow to the huge size that we can foresee in real world future applications. Many different approaches exist to tackle ontology modularization, corresponding to different goals or building on different initial hypotheses. This chapter aims at clarifying the vision of the domain by providing a detailed yet generic characterization of the issues and solutions related to the various facets of modularization.

1.1 Introduction

Complexity is an almost pervasive feature of modern life. Computerized knowledge management is no exception. We, both as individuals and as societies, increasingly rely on knowledge stored in computers worldwide to acquire the pieces of information we need to govern our personal behavior. From politicians to technical directors, managers in charge of societal problems (e.g. environmental issues) similarly rely on accurate extraction and synthesis of relevant knowledge. Unfortunately, knowledge repositories have grown beyond our capacity to apprehend their content. We are constantly exposed to knowledge overdoses, and one of the critical success factors nowadays is the capability to get rid of all knowledge that is unnecessary to the task at hand, so that the knowledge to be considered downsizes to a manageable amount. Accordingly, the research community is increasing its effort towards the specification of frameworks and techniques that may help in downsizing knowledge.

Modularization is one of the approaches to achieve such a result. In its most generic meaning, it denotes the possibility to perceive a large knowledge repository (be it an ontology or a database) as a set of modules, i.e. smaller repositories that, in some way, are parts of and compose the whole thing. Modularization materializes the long-established complexity management technique known as divide and conquer. It is routinely used in various areas of computer science, such as algorithms, software engineering, and programming languages. In software engineering, for example, module is one of the terms used to denote a software component that is designed to perform a given task and is intended to interact with other modules within a larger software architecture. In programming languages, module sometimes denotes an encapsulation of some data. Easiness of understanding and potential for reusing are among the main claimed benefits of these approaches.

In the knowledge management domain, modularization opposes the initial and traditional vision of databases as an integrated and monolithic collection of strongly interrelated data. This explains that the database community basically discarded modularization. Even federated databases were seen as a way to build larger databases, not as a way to address complexity. Only few researchers (e.g., [20] [17]) addressed for example the issue raised by the complexity of handling database schemas having a very large number of object and relationship types. The work in [20] proposes design techniques supporting multiple levels of details (i.e. more or less detailed representations of the database schema), while the work in [17] develops a design methodology built on the idea of manually identified design units, such that each unit covers a part of the application domain.

In contrast, concerns about scalability and interoperability of ontologies have generated a significant interest in modularization from the semantic web community. Ontologies are the knowledge repositories that enable meaningful inter-agent communication in semantic web frameworks. To support the huge number of services that will be developed in the semantic web, ontologies will grow larger and larger to accommodate as much knowledge as possible. The universal, all-encompassing ontology is the dream behind this trend. In more realistic approaches, ontologies are targeted to cover a specific domain. But even these domain ontologies may grow in size beyond human mental capacity and pose a critical challenge to reasoning tools. An example is the ProPreo (Proteomics data and process provenance) ontology, which in 2006 contained 18.6 million assertions for 390 classes and 3.1 million instances [16]. Moreover, the domain covered by an ontology may be expanded to be of use to a wider community and to promote interdisciplinary research. For example, an ontology for biology (e.g. the Gene ontology [18]) may eventually combine with an ontology for medicine to provide for better interoperation of specialists in the two domains. A rising number of instances may be the growing factor for an ontology used to cover an increasing number of similar applications in a given domain. For example, ontologies for tourism will grow beyond limits if they are meant to hold instances about any possible tourist destination worldwide. There clearly is a scalability issue, and modularization is an appealing solution as it aims at replacing oversized ontologies with smaller ones. The issue originates in both growth in concepts and growth in instances, although the two phenomena do not raise exactly the same issues, and may call for different solutions.

Conversely, there is also a definite need from applications to gather knowledge from several, not just one, ontological sources. It is known that, when knowledge is distributed, the idea to collect all knowledge into a single repository (i.e. the integration approach) is very difficult to implement, because of semantic heterogeneity calling for human processing. Instead, implementing an interoperability framework that supports meaningful knowledge exchange among the sources is an easier and more efficient approach. The definition of such a distributed collaborative paradigm that enables collaboration among existing ontologies (now seen as modules of the larger virtual ontology that is built by the collaboration and corresponds to application requirements) is a challenging research stream that comes under the umbrella of modularization and has already generated several competing approaches.

The chapter hereinafter continues with the overview of modularity ideas. The next section discusses goals that may be assigned to modularization. A given goal may influence the way the concept of module is defined. Section 1.3 focuses on the concept of module per se. Section 1.4 looks at the strategies that may be used to produce modules using semantic or syntactic criteria, with human-driven or automatic approaches. Section 1.5 discusses the correctness criteria that should characterize modular ontologies. Section 1.6 analyzes issues in composing existing ontologies as modules for larger repositories. Module interconnection, essential to distributed reasoning, is discussed in Section 1.7. Section 1.8 introduces multi-perception features. Section 1.9 concludes the chapter.

1.2 Goals of Ontology Modularization

Modularization in itself is a generic concept that is intuitively understood as referring to a situation where simultaneously a thing (e.g. an ontology) exists as a whole but can also be seen as a set of parts (the modules). How modularization is approached and put into practice, as well as what are the advantages and disadvantages that can be expected from modularization greatly depend on the goals that are pursued through modularization. This section briefly reviews some frequently quoted possible goals for modularization of ontologies. A complementary discussion, in Chapter 3 of this book, focuses on measurable goals that can be used as evaluation criteria to assess the relative quality of modularization techniques versus expected benefits.

Scalability for querying data and reasoning on ontologies

This is an all-embracing goal, which mainly sees modularization as a way to keep performance of ontology services at an acceptable level. Performance concerns may be related to query processing techniques, reasoning engines, and ontology modeling and visualization tools. Database systems have basically solved performance issues by developing efficient storage and querying optimizers (e.g. dynamic hashing and multidimensional indexing) to provide fast retrieval and update of data. As they do little reasoning, this task is not problematic for DBMS. Instead, for ontological reasoners the complexity of their task is directly related to the number of axioms and facts they have to explore before coming to a conclusion. While different reasoners are available, supporting different formalisms from RDF to OWL, they are known to perform well on small-scale ontologies, with performances degrading rapidly as the size of the ontology (in particular, the number of instances) increases. Keeping ontologies smaller is one way to avoid the performance loss, and modularization is a way to replace an ontology that tends to become oversized by smaller subsets. Modularization fulfills the performance goal if, whenever a query has be evaluated or an inference performed, this can be done by looking at just one module, rather than exploring the whole ontology. But if most queries call for distributed reasoning to exploit knowledge from several modules, it remains to be demonstrated that the overall time for coming up to a result is less than the time required for the same task executed against a single ontology holding all the knowledge kept by the modules in the network. Thus, the driving criterion for modularization aiming at performance is to localize the search space for information retrieval within the limits of a module. Implementing a good distribution of knowledge into modules requires knowledge about the search queries that are expected. This type of knowledge can be extracted a posteriori from observing information queries over some period of time. Predicting this knowledge a priori would be more effective, but is difficult to achieve.

Scalability for evolution and maintenance

Achieving this goal is an open issue for both ontologies and databases. Here, the driving criterion for modularization is to localize the impact of updating the knowledge repository within the limits of a module. Implementing a good knowledge distribution requires an understanding of how updates propagate within the repository. It also requires knowledge on the steadiness of the information it contains. Steadiness here is meant to denote the unlikeliness of an evolution. A possible factor for steadiness is the confidence level attached to information. How confidence levels, and more generically steadiness indicators, are acquired remains an open issue for research.

Complexity management

While scalability usually refers to system performance in processing user queries, performing reasoning tasks, and visualizing results, a similar issue can be raised regarding the design of the knowledge repository. The larger the repository is, in terms of objects and relationships or in terms of axioms and rules, the more difficult is ensuring the quality and accurateness of the design, especially if the designers are humans (as it is still the case). We refer here to the issue of making the design task intellectually affordable for designers, thus creating the best conditions leading to a semantically correct design that fulfills the informational requirements assigned to the ontology/database at hand. Quoting Stuckenschmidt and Klein [15], ”ontologies that contain thousands of concepts cannot be created and maintained by a single person”. The best way to help designers is to reduce the size of the design problem. That is what modularization achieves. Let designers design modules of a size they can apprehend, and later either integrate these modules into the final repository or build the relationships among modules that support interoperability. This is a typical application of the divide-and-conquer principle.

Notice that the facilities provided by DL reasoners to check the consistency of specifications address a different issue: correctness. These facilities are essential to guarantee the quality of a given design, but they only lift a specific problem from the concerns of the designer. They do not make the design task easier.

Understandability

Another challenge is to be able to understand the content of an ontology (and of the schema of a database), an obvious prerequisite to the ability to use them. This is very similar to the previous challenge, but has to do with the usage phase rather than the design phase. Whether the content is shown in visual or textual format, understanding is easier if the repository is small, for example just a module. Smaller repositories are undoubtedly preferable if the user is a human being (as is visual versus textual representation). We believe intelligent agents navigating through the Web-services space also benefit from smaller repositories. The agent might have to explore (browse) the schema or ontology rather than just querying it. Browsing a few nodes is faster than browsing a huge number of nodes. Size, however, is not the only criterion that influences understandability. The way it is structured contributes to improving or decreasing understandability, as it has been extensively shown in the database modeling domain (i.e., a relational schema is much harder to understand than its equivalent entity-relationship or UML schema).

Context-awareness and Personalization

Context is a pervasive concept that also applies to knowledge repositories, and personalization is a specific kind of context use driven by the user profile. Efforts towards context-dependent databases and ontologies are being pursued. Context-awareness means that the database/ontology system knows that different subsets of its content are relevant for different contexts. This implies that the creation of knowledge, its storage and usage have to take context into account. What exactly makes up a context is still an open issue and there is no universal answer to the question. Contexts are themselves context-dependent, i.e. they convey the specific requirements of the applications using the contextual knowledge. One particular kind of context is ownership of information, known to be an important factor to be taken into account when organizing distributed cooperative systems. This may also apply to ontologies, although most of them are seen as publicly available resources. Ownership in these cases provides the rationale for building a modular ontology. Ownership information can also be attached to existing ontologies in view of integrating them into a modular ontology where modularity is based on personalization.

Reuse

Reuse is a well-know goal in software engineering. Reuse is most naturally seen as an essential motivation for approaches aiming at building a broader, more generic repository from existing, more specialized repositories. However, it may also apply to the inverse approaches aiming at splitting a repository into smaller modules. In this case, the decomposition criterion would be based on the expected reusability of a module (e.g., how well can the module fill purposes of various applications?). Reusability emphasizes the need for rich mechanisms to describe modules, in a way that maximizes the chances for modules to be understood, selected and used by other services and applications.

1.3 The Essence of Modules

Ontologies and databases are not arbitrary collections of concepts/objects, relations, properties, axioms and instances. The gathering and definition of their elements fulfills a purpose, be it to make available to potential users the knowledge and terms covering some domain or to represent a subset of the real world that is of interest to an organization and its applications. Similarly, the process of decomposing a (database or ontology) repository into modules has to rely on some meaningful principles to produce modules that make sense. Each module is expected to show a similar unit of purpose, gluing together those elements from the global repository that participate to a given goal (which may be seen as a sub-goal of the goal of the global repository). For example, an ontology module would represent an agreed conceptualization of a sub-domain of the domain of the ontology [3]. Such modules make sense for, and can be separately used by the community of users only/mainly interested in the sub-domain rather than the whole domain covered by the overall ontology. For example, an enterprise database or ontology may include modules for payroll, for customer support, for accounting, for marketing, etc. An ontology for historians may have modules for chronological subsets (e.g., pre-history, ancient, medieval, renaissance, and contemporary periods) and at the same time have modules by geographical area (e.g., European, American, Asian, Middle-East, African, Oceanic history).

For knowledge management consistency, a database module shall be a (smaller) database, and an ontology module shall be a (smaller) ontology. The benefit is that the same software system can handle the whole repository and its modules. From a pragmatic viewpoint, modules of the same database or ontology are most likely defined and used according to the same model and formalism (e.g., they all follow OWL-DL specifications or they all are relational databases), but this is not a mandatory feature. The same rationale applies when a module is built by extracting some content from the ontology/database, without running a global decomposition process.

The potential specificity of modules, making them different from the whole, is that a module is aware of being a subset of a broader knowledge organization, and therefore knows it may interact with other modules to get additional knowledge in a sub-domain closely related to its own sub-domain. Thus, on the one hand (see Part II of this book) each module interacts with the whole in a part/whole relationship (leading to a focus on extraction techniques), while on the other hand (see Part III of this book) modules may have a collective societal behavior, where the focus is on cooperative techniques. Using an arithmetic metaphor, we can summarize this as:

module = a (smaller) ontology + inter-modules links

The composition/interrelation duality supports two different contexts that people associate with the idea of modularization. In one context modularization refers to the process of turning several self-standing repositories into either a collection of interrelated and interoperating modules that together form a broader repository (cooperative or distributed approaches), or a single integrated broader repository (integration approaches). In this context the modules are simply the existing repositories. The issue is not how to delimit them, but to enable their interoperation or integration. In the other context, modularization refers to the process of creating one or more modules from an existing repository. Many different ways to do that have been proposed. The next section discusses the issue.

1.4 Modularization Strategies

Defining the strategy and assumptions that rule how knowledge in a repository is distributed into modules is a fundamental task in approaching modularization. Strategies may rely on semantic or syntactic criteria, informal or formal specifications, vary in degree of automation and vary in terms of targeted goal. The latter includes the issue whether the coexisting modules created from a given ontology have to be disjoint or may overlap. This section first discusses disjointedness, then moves to examining the different strategies.

Disjoint or overlapping modules

One of the basic alternatives in fixing a modularization strategy is whether the strategy should enforce disjointedness among the modules of an ontology, or allow them to overlap. The main advantage of disjointedness is easier consistency management. Contrary to overlapping modules, disjoint modules cannot contradict each other. Hence a set of consistent disjoint modules is always consistent. Enforcing disjointedness entails that distribution of knowledge into the modules has to be controlled by the system. The system may have complete control, meaning that it automatically performs distribution of knowledge using some partitioning algorithm. Alternatively, users may somehow fix the distribution, in which case two options exist to enforce disjointedness. One option lets the system check that modules are kept disjoint by users (disjointedness is explicitly enforced), rejecting knowledge insertions that violate disjointedness. In the other option whenever the user allocates a given piece of knowledge to multiple modules, the system creates as many copies as needed to store one in each targeted module and does not retain the fact that copies exist (disjointedness is implicitly assumed). Otherwise stated, the system iterates the insertion but ignores the resulting duplication (each copy is considered as a separate and unique piece of knowledge).

On the other hand, the advantage of overlapping is more flexibility in knowledge distribution decisions and more functionalities in using modules.Assume, for example, that a modular ontology about *History* is required. Possible targeted modules could include a module about *Middle Age* and another one about *Italy*. Clearly, a number of concepts, e.g. the *History of Medieval Italy* concept, will be relevant for both these modules. Each of these concepts is candidate for allocation to multiple modules. In a disjointedness-oriented strategy, one may simply create as many copies of a concept as needed to allocate one copy to each candidate module, and then forget about the duplication, i.e. consider each copy as a separate piece of knowledge, independent from the other copies of the same concept. In an overlapping-oriented strategy, concepts may be directly allocated to multiple modules (without being duplicated) and the system keeps awareness of this multiplicity. Using this awareness the system can let users navigate from one module to another, i.e. from one instance of a concept in one module to another instance of the same concept in another module.

Issues related to such knowledge overlapping form a research domain per se. Practically overlapping modules may be implemented in two different ways. The modules may be created to share the same interpretation domain, as that may be the case of ontologies and databases created as modular since the beginning, like Mads (see Chapter 5 in this book). Alternatively, each module may have its own interpretation domain, and these domains be linked, two by two, by binary relations, like in distributed ontologies (see Chapter 12 in this book). Remark that when real world entities are described at the same level of granularity in two modules, the binary relation should reduce to an injective function. A "full" binary relation allows mapping an instance (object) of a module to a set of instances (objects) of the other module, which is very different from a simple overlap.

Semantics-Driven Strategies

The most intuitive approach to modularization is to let it be driven by the semantics of the application domain and the way the application perceives identifiable subdomains which could stand by themselves and handle a subset of the application requirements. This has been stated as a "semantic correctness" requirement for modularization: "A module should make sense" for the users [7].

Semantic interpretation of knowledge relies on human appreciation, i.e. on the expertise (knowledge of the domain covered by the ontology and knowledge of application requirements for a database) of the persons in charge of creating and updating the content of the repository. The role of the system is usually limited to recording the allocation of knowledge items to the modules, ensuring that every single content component (e.g., concept, role, axiom, and instance for ontologies) is allocated to at least one module (assuming the repository is fully distributed over its modules). Modular ontology approaches such as Cyc and Mads (see Chapter 5 in this book) follow this path. In modular ontologies the specification of a new piece of knowledge includes its allocation to modules. We can say these are manual strategies, as modularization is crafted piece by piece by users. Cyc nevertheless offers tools for automatic placement of a new assertion in the most relevant module, as determined by examining the concepts in the assertion. This assistance to users may help in avoiding erroneous distribution of knowledge, such as allocating a concept to a module and its properties to another module. Some proposals address this concern through the definition of semantic units that group knowledge into atomic pieces whose components are likely to be kept together when defining the distribution into modules. For example, Rector [12] proposes a semantic reorganization ("normalization") of the hierarchy of concepts in order to lead to a more semantic partition of the ontology. This normalization is proposed as a preliminary to the extraction of modules from the ontology.

Some approaches, where ontology and module creation are desynchronized, are also driven by semantics and aim at bringing more automation into the module elaboration process. It is the case of most methods that extract modules out of a classic, non modular, ontology. The idea of these semi-automatic approaches is simply to compute the desired modules from initial specifications provided by users. The human designer role is to specify the desired result and the system role is to actually build the result. A simple and quite traditional way to specify the desired module is by writing a query that selects a subset of the ontology. The evaluation of the query produces the module, extracted from the ontology [21] [9]. The newly extracted module is materialized, and is independent from the ontology.

Structure-Driven Strategies

Other modularization strategies purposely avoid considering semantic aspects. They look at an ontology as a data structure consisting of a graph of interconnected nodes (whatever their semantics) and use graph decomposition algorithms to create subsets (the modules) based on structural properties that a subset has to comply with. For these strategies, a module is a set of concepts (and roles) that are more tightly inter-related together than to other concepts that are not in the set. What tightly interrelated exactly means varies from one proposal to another one. Hence different strategies belong to this category, each one characterized by the specific structural criteria they use for the decomposition. A significant example is Chapter 7 in this book, which presents in detail a structure-based algorithm that produces a partition of an ontology into modules. Such semantic unawareness allows the algorithm to run automatically without relying on human input. This is essential whenever the goal is to allow computerized agents to produce ontological modules without calling for human interaction.

However, these algorithms can also be tuned by initially specifying the application requirements that may influence the decomposition criteria. Typically, the module designer identifies within the ontology a small set of kernel elements (i.e., the concepts and roles most important to the new module) and in a second step the system iteratively builds a module from the kernel elements by adding other elements attached to the selected ones. This kind of "growing" algorithm needs some criterion to determine how far it should go in looking for attached elements (e.g., using something like a threshold for distance between the kernel elements and the other elements). Moreover, a strategy has to be defined to instruct the system on what to do whenever two related elements are allocated to different modules. Strategies presented in Chapters 8 and 9 build on the concept of kernel elements and associated structural criteria to automatically extract modules from an ontology.

In the database domain, fully automatic decomposition has been investigated, for example, to allocate data within a distributed database framework based on load balancing algorithms, i.e. a criterion driven by system performance concerns. The algorithms produce a data distribution schema that shows which data fragments have been defined and where they are stored [4].

A different category that may also be classified as structure-driven groups the strategies to build a modularized ontology from existing ontologies considered as modules of the targeted all-embracing ontology. The structural criterion on which modules are defined is simply to take the existing pieces as modules and interconnect them to form the new ontology. One could say that modules pre-date the ontology, while the other strategies derive modules from the ontology. Representative of these strategies are distributed ontologies (as discussed in Part III of this book) and federated databases [13]. Strategies differ from each other on the techniques they use to interrelate the module (see Section 1.7 hereinafter).

Machine Learning Strategies

An alternative to human-driven modularization, also not needing an explicit modularity criterion, is computed modularization based on some machine learning knowledge acquisition process. For example, an analysis of the queries that are addressed to the repository and of the paths within the repository that are used to respond to queries may be used, considering the frequency of paths and their overlapping, to determine the clustering rule that produces the optimal decomposition. We use the term machine learning in a very loose sense including any one of the many available techniques that a system may use to infer knowledge from any data log. Data mining and clustering analysis are candidate techniques. These strategies differ from previously discussed strategies in the sense that the knowledge distribution they produce is purely based on observing interactions that are relevant for knowledge management, not on a human or human-driven dedicated specification (as in semantics-driven strategies), nor on structural properties of the ontology (as in structure-driven strategies). To the best of our knowledge, machine learning strategies have not yet been explored for ontology modularization.

Monitoring Modularization and Making it Evolve

At last, we mention here the problem of updating the distribution of knowledge into the modules, irrespectively of the strategy used to create them in the first round. Whatever the modularization strategy, it is important to evaluate and monitor the reliability and efficiency of the knowledge services provided by modules. Knowledge distributions that, once in use, prove to be unsatisfactory or inefficient shall be adjusted or corrected based on the outcome of monitoring tasks. The outcome may for example identify a subset of a module that is frequently needed by another module, thus suggesting that this subset should more properly be allocated to the other module. Similarly, unused parts of modules may simply be deleted or transferred to an archival module. Whenever queries to a module always require additional knowledge from another module, merging of the two modules might be advisable. Conversely, if query patterns show independent usage of separate parts of a module, a splitting of the module is worth considering.

1.5 Module Correctness

A very important concern when decomposing a repository is the correctness of the modularization. A basic requirement for a modular ontology to be correct is that each module is correct.

To evaluate correctness, first we have to define what the term means. An obvious correctness criterion is syntactic correctness, that is to say the knowledge specifications follow the rules of the language and data model they rely on. Enforcing this kind of correctness is usually done by the underlying tools, e.g. ontology editor tools and schema design tools for databases. At the instance level, checking the consistency between instances and specifications is done by the reasoners and the DBMS.

In the previous section we quoted the definition by Grau et al. of semantic correctness as referring to the fact that a module should make sense. Obviously there is no way for a system to check such semantic correctness, in particular in the ontology world where ontologies (unlike most of databases) are not confined to hold the knowledge required by a specific application.

The same authors also defined the logical correctness concept, where logical refers to the use of logic. Their definition states that a module is logically correct if and only if, as long as it involves only the vocabulary of the module, the knowledge that can be inferred by the module is the same as the one that could be inferred by the initial ontology (see Chapter 6 in this book).

The links that interrelate the modules of an ontology must be taken into account when computing the inferences generated by a module. This means that knowledge targeted by the links should exist (no pending references) and be consistent with the knowledge inside the module, i.e. augmenting the knowledge within the module with the knowledge external to the module should not result in unsatisfiability. However, recent work shows that a certain level of localized inconsistency between external and internal knowledge can be allowed in the sense that queries that do not involve the inconsistent subset can be evaluated with no problem [2].

In case an existing ontology is split into modules by partitioning, the collection of modules should satisfy specific correctness criteria that assess the preservation of information in the partitioning process. Information preserving may be defined as the fact that the result of any query addressed to the collection is logically the same as the result of the query addressed to the original ontology. This is sometimes referred to as the fact that the whole ontology is a conservative extension of each one of its modules. Chapters 2 and 6 in this book build on this conservative extension idea to formally define properties of modules generated using alternative modularization schemes. This allows ontology designers to choose a modularization approach that best matches the properties they want to be achieved.

Information preserving can also be statically defined as the fact that, when re-composing the original ontology from the modules (using some composition rules), what is obtained is equivalent to the original piece, nothing less, and nothing more[1]. Depending on the modularization rules used, it may be possible to guarantee that, if rules are obeyed, the modularization they generate is information preserving. If information loss cannot be avoided, an estimation of the information loss can be a very useful add-on for the query answering techniques [8].

1.6 Ontology Composition

The idea to collect data/knowledge from several existing repositories to form a new, real or virtual repository designed for new applications has arisen as soon as computer networks have become operational. Current economic trends have turned the idea into a necessity. In the database domain the idea has become very popular in the research community in the 80'ies and 90'ies under the label "federated databases" [13], denoting new databases built by integrating sets of existing databases. The main research issue from the semantic viewpoint has been data and schema integration, i.e. how to build a homogeneous description (the integrated schema) of the whole set of data available in the existing databases, despite their syntactic and semantic heterogeneity [10][14]. Later, the trend turned to alternative solutions that would support integrated data management without needing an integrated schema. These approaches are frequently denoted as cooperative systems. Eventually, new cooperation paradigms have emerged in the framework of peer-to-peer systems[2]. As the name says, in these frameworks the component database systems (the peers) directly interact with each other whenever they need information they do not have. Different strategies use different heuristics for a peer to determine which other peers to contact, given that no central description is available to tell which peer holds the desired information. The latest trend build on memorizing the results of these exchanges to gradually build in each peer knowledge of where relevant information has been found. This trend is known as emergent semantics approaches [1][5].

In the ontology world, the same two approaches exist: Integration of existing ontologies into a global, classic, non modular, ontology is similar to the federated databases approach. On the other hand, distributed ontologies is similar to cooperative databases systems. A distributed ontology is a broader global ontology, covering a larger domain, which is built by somehow "composing" a set of existing ontologies. In this setting, from the modularization viewpoint the existing ontologies are pre-existing modules used to build a modular ontology. The goal is not to create a single big ontology (an effort that would probably fail for the same heterogeneity issues that made data integration into federated databases a dream rather than a reality) but to target a new modular and cooperative ontology, where the modules (existing

[1] This is analogous to information preserving rules in databases for the decomposition of a relational relation into a set of relations of higher normal form (known as the normalization process).

[2] `http://en.wikipedia.org/wiki/Peer-to-peer`

ontologies) are kept unchanged and acquisition of new knowledge is done within the modules rather than at the global ontology level. Part III in this book presents the best established composition approaches.

The process of composing the modules consists in identifying duplicated and complementary knowledge among modules, and then establishing appropriate links in between the identified pieces of knowledge. Duplicate knowledge can be linked using a multi-instantiation link, i.e. a link whose semantics says that the two linked elements contain each one a subset of instances that represent the same set of real world entities; or it can be linked using a "same as" role, whose semantics says that two different instances are in fact two representations of the same real world entity. Complementary knowledge is linked using inter-module roles, i.e. roles whose domain is in one module and whose range is in another module. For example, a Person concept in module M_i can be linked by a role to the Car concept in module M_j assuming that the intended semantics is that a person may own a car.

It is also possible to devise strategies to compose existing ontologies where some rules are used to redistribute or reallocate the content of modules within the global ontology. For example, the strategy could instruct the ontology system to remove duplicate content and replace it by inter-module links, in an effort to reduce the individual or the cumulative size of the modules. However, this would be against the autonomy of the modules, and should therefore applied with care, if ever.

As in federated databases, a typical issue in module composition/integration arises from the syntactic and semantic heterogeneity that may exist among multiple descriptions of the same concepts and roles in the different modules. Syntactic differences include heterogeneity of the underlying formalisms (e.g., one module uses a paradigm from logics while another module uses a conceptual data modeling paradigm), and of the languages used to define them (one module uses RDF, another one uses OWL-DL). Differences in representation of shared elements arise whenever the sub-domains covered by the modules are not mutually disjoint, a situation that is likely to be the norm rather than the exception. For example, one module holds road network knowledge while another one holds knowledge on roads as part of a public transport system, resulting in different representation requirements of roads in the two modules.

The whole ontology matching process has been and still is extensively discussed, e.g. in [6]. Part of the issue is the detection and processing of the semantic differences that otherwise obscure knowledge sharing. Indeed, corresponding elements, i.e. elements that at least partly represent the same set of real world entities or links, may differ at the terminological level (different names), descriptive level (different sets of properties), structural level (different constructs of the model are used to represent the elements), and semantic level (the elements describe different sets of real world entities or links). The process to detect the correspondences between the elements is referred to in the integration literature as similarity search or correspondence discovery.

When the goal is to create a global repository (as in federated databases) with a single description (the integrated schema), differences between the source descriptions have to be reconciled to enable generating the integrated element. This reconciliation process is called conflict resolution. Similar reconciliation is nowadays proposed for integration of multiple ontologies into a single ontology. Integration research is not discussed in this book as its goal is opposite to modularization. Instead, distributed ontology approaches fully qualify as modularization techniques, and are reported in detail in Part III of this book. In distributed ontologies the only task is to put an inter-modules link between the corresponding elements. Terminological and descriptive differences do not matter. But the inter-modules links should support structural and semantic differences. They should allow to link elements of different kinds and they should express various kinds of semantic differences, i.e. the represented sets of real world entities/links are the same, or one is the subset of the other, or the two sets are overlapping, or the two sets are disjoint but related.

1.7 Links among Modules

In a modular ontology, each module is assumed to provide ontological services related to a given sub-domain of the global domain covered by the ontology as a whole. Basic experience in knowledge organization shows that very frequently it is impossible to partition a domain into disjoint sub-domains. Whatever the modularization principle is, some concepts inevitably end up belonging to multiple sub-domains. Most frequently, the way a concept is characterized changes from sub-domain to sub-domain. Usually, some characteristics, referred to as inherent or essential to the concept, are the same whatever the sub-domain, and many others, referred to as incidental to the concept, are specific to the sub-domain they relate to. This means that complementary and duplicated knowledge about a concept exists in different modules. Users of a module may be interested in accessing the entirety of the knowledge about a concept available within the modular ontology. Similarly, given the availability of two modules, users of one module may be interested in connecting concepts of this module to other concepts in the other module whenever they see a meaningful and useful connection between the two. To gain direct access to knowledge external to a module, inter-module links are established. They create a navigational path from module to module that makes the knowledge in the target module available to users of the initial module.

Inter-module links may be seen as an integral part of the module they stem from, or as a construct that is superimposed onto the modules and external to them. In the first case a module per se is an incomplete ontology given that its links to external elements appear as pending links when the module is considered in isolation, for instance for consistency checks. In the second case, links can be considered as externally stored, possibly in a separate dedicated repository, and each module taken in isolation is an autonomous ontology. This solution has for example been adopted in distributed database systems, where tables with pairs of oids have been used to store the instance-level correspondence between related objects in different databases. Managing links externally to modules allows creating different ontologies from the same set of modules by defining alternative sets of links.

Inter-module links may be of different types: relationships or multi-instantiation links. Inter-module relationships may relate two concepts of two different modules, concepts that describe different sets of real world entities that are related by a relationship. For instance, one module may describe cars and another one models of cars. An inter-module relationship or role, e.g. hasModel, can be defined between the two concepts. Its instantiation will have also to be defined. Chapter 11 presents such a solution based on the $\mathcal{E}$–connections technique which is a method for combining logical languages and in particular OWL-DL ontologies. The ontologies are related by inter-module roles that link concepts of two different modules. The method supports an extended version of the Tableau algorithm for reasoning on distributed ontologies of kind $\mathcal{E}$-connections.

Inter-module multi-instantiation links are links that relate two elements that belong to different modules and describe, at least partially, the same set of real world entities or links. The most usual kind of inter-module multi-instantiation links is the is-a link between two concepts, for example Car in one module and Vehicle in another one. C-OWL calls this kind of link, bridge rules. More generally, an inter-module multi-instantiation link may express sub-set (is-a), equivalence or overlapping. Chapters 5 and 12 support this kind of links. Notice that the linked elements may be two concepts, two roles, a concept and a data-type, and a composition of roles and a role.

In most distributed ontology approaches, inter-module multi-instantiation links relate elements of the same kind, i.e. two concepts, two roles or two individuals. But more complex links are needed, for instance a role may correspond to a composition of roles (e.g. *isUncle* corresponds to *isParent.isBrother*), or a concept may correspond to the union of several concepts (e.g. *Parent* corresponds to $Father \cup Mother$).

As in distributed ontologies each module is an autonomous ontology, it has its own interpretation domain, which is disjoint from all the other ones. Therefore, inter-modules links of kind is-a, or more generally multi-instantiation links, require a mapping between the interpretation domains of any two linked modules. In most of the approaches the mapping is defined by a binary relation. That allows to assert that an instance of one module corresponds to any number of instances of the other module. Consequently, the semantics of an inter-module is-a link depends on the underlying binary relation. For instance in Chapter 12, a bridge rule $O1{:}C1 \xrightarrow{\sqsupseteq} O2{:}C2$ does not always say that any instance of *C1* corresponds to an instance of *C2*. If the binary relation leaves out some *C1* instances, that would not be the case: The bridge rule constrains only the instances of *C1* and *C2* that participate in the binary relation.

When a distributed ontology contains inter-modules multi-instantiation links, an important question is the consistency of the related modules. As already discussed in Section 1.5, knowledge sharing by different modules is prone to inconsistencies (unless a tight control and coordination of the evolution of the modules are enforced). This calls for specific consistency-preserving mechanisms. A distributed ontology will be consistent only if each component ontology is itself consistent and all the multi-instantiation links are consistent with the component ontologies they are linking. For more general distributed reasoning issues see Chapter 12 in this book.

Another facet of the consistency issue about links is how up-to-date they are. Modules, as ontologies in general, are expected to evolve in a desynchronized and decentralized way. If a link has been identified and described at time t, can we guarantee that it is still valid for use at some later time $t' > t$? For example, assume a link specifying that a concept A in module M_i is equivalent to a concept B in module M_j. The evaluation of a query to M_i that references A may need to traverse the link between M_i and M_j to find out what M_j knows about B. If, after the link has been established, a role r(B, D) is added to M_j which somehow constrains B, it may be that the new constraint results in the fact that the equivalence does not hold anymore. To prevent inconsistencies, either an update propagation mechanism is developed to keep all links up-to-date anytime, or links have to be invalidated whenever the semantics of one of the elements they link changes.

1.8 Contextual Modules

As discussed in section 2, irrespectively of issues related to the size and complexity of knowledge repositories, modularization may be used as a means to achieve contextualization and personalization of knowledge services. Modules in this perspective are not conceptualizations of different sub-domains but different conceptualizations of the same domain, each one tailored for a specific usage context. Each conceptualization stems from a specific perspective on and interpretation of which knowledge is relevant and how it has to be represented according to many parameters, including the observer's background and preferences, the purpose assigned to its representation, the temporal and spatial framework of the observation. For example, should the spatial context be North America and the temporal context be before the first arrival of Europeans, the concept of *horse* would not be part of the relevant knowledge as the concept was not known to American Indians at that time. The concept become relevant with the arrival of Europeans, and denotes a kind of quadruped. For a more recent temporal context, the term would also denote heroin (in slang). Later, the same term would also denote persons smuggling drugs. Notice that recording synonyms and other lexical relationship among words, as done in terminological ontologies (e.g. Wordnet), does not suffice to convey the complexity of contextual influence on knowledge description. Context-dependent modules for the same ontological domain may provide the means to achieve context-aware information services. These modules may be independent from each other, but most likely they will have interdependencies to the extent they share some common knowledge. In particular, one module may include another module [5]. For example, all knowledge about a university system according to a "Professor" context may also belong by definition to a more generic "Academic" context for representing the same university system. We would then say that the Academic context includes the Professor context.

This approach to modularization suggests that the contexts of interest have been defined first, so that when entering new knowledge into the repository the context(s) to which it belongs can be readily identified. Existing context-awareness proposals are built on this hypothesis [5][19][11]. They are discussed in Chapter 5 of this book. A posteriori allocation to contexts is possible but is likely to call for manual annotation, usually a non-affordable task.

1.9 Conclusion

Managing large ontologies is a serious challenge for ontology designers, reasoners and users. One generic strategy to deal with the problem is modularization, which aims at replacing a huge ontology by a collection of smaller component ontologies, called modules, that can be manipulated independently from each other and are nevertheless capable of collaborating in providing the same service as the whole initial ontology. Although knowledge modularization is mainly a relatively new research domain, it has attracted the attention of several research groups and workshops trying to deal with large and distributed ontologies. Many research proposals have therefore been documented in the literature. Tendencies have appeared and enable an organized survey of current achievements, which is the subject of this book. Thus, most significant achievements related to the topic are described in detail within the book.

As a preamble, this initial chapter developed a global analysis of modularization issues, caring to take into account the various facets and perceptions of ontology modularization and suggesting possible answers to the many open problems. In particular, we identified a composition versus a decomposition approach to modularization. In the former, a set of existing source ontologies are apprehended as modules of a larger ontology that is built from the preexisting sources using some integration or cooperation technique. In the latter, it is the global ontology that pre-exists, and modularization is seen as the process of producing a consistent set of sub-ontologies, the modules, using some decomposition technique. Beyond this major split within the set of approaches that deal with modularization, we identified a number of issues, from the precise definition of the module concept to how different modules can be interconnected to benefit from complementarities of their semantic content. This generic analysis is further explored in following chapters 2 to 4 in this Part I of the book, focusing on formal properties of modularization and evaluation criteria for modularization techniques, as well as the variety of knowledge importing techniques that can be used to transfer knowledge between modules.

We have focused on identifying and showing alternative approaches, with their underlying assumptions as well as with their specific goals. The need for modularization does indeed emerge from different contexts, characterized by different requirements. A multiplicity of solutions is required to cover all potential useful contexts.

However, neither this chapter nor the book exhaust the possible research avenues that remain open for the future. For example, one direction for future work is to focus on using fuzzy techniques to support the possibility to attach different confidence degrees to the mappings between ontological modules, thus leading to some form of fuzzy modular ontology.

References

1. Aberer, K., et al.: Emergent semantics principles and issues. In: Lee, Y., Li, J., Whang, K.-Y., Lee, D. (eds.) DASFAA 2004. LNCS, vol. 2973, pp. 25–38. Springer, Heidelberg (2004)
2. Bouquet, P., Giunchiglia, F., van Harmelen, A., Serafini, L., Stuckenschmidt, H.: Contextualizing ontologies. Journal of Web Semantics 1(4), 325–343 (2004)
3. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: DL-lite: Tractable description logics for ontologies. In: Proceedings of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005) (2005)
4. Ceri, S., Pelagatti, G.: Distributed Databases - Principles and Systems. McGraw-Hill, New York (1984)
5. Cudre-Mauroux, P., Aberer, K. (eds.): Viewpoints on emergent semantics. Journal on Data Semantics VI, 1–27 (2006)
6. Euzenat, J., Shvaiko, P.: Ontology Matching. Springer, Heidelberg (2007)
7. Grau, B.C., Parsia, B., Sirin, E., Kalyanpur, A.: Modularity and web ontologies. In: Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR 2006) (2006)
8. Jarrar, M.: Modularization and automatic composition of object role modeling (orm) schemes. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM-WS 2005. LNCS, vol. 3762, pp. 613–625. Springer, Heidelberg (2005)
9. Magkanaraki, A., Tannen, V., Christophides, V., Plexousakis, D.: Viewing the semantic web through RVL lenses. Journal of Web Semantics 1(4), 359–375 (2004)
10. Parent, C., Spaccapietra, S.: Database integration: an overview of issues and approaches. Communications of the ACM 41(5), 166–178 (1998)
11. Parent, C., Spaccapietra, S., Zimányi, E.: Conceptual Modeling for Traditional and Spatio-Temporal Applications: The MADS Approach. Springer, Heidelberg (2006)
12. Rector, A.: Modularisation of domain ontologies implemented in description logics and related formalisms including OWL. In: Proceedings of the K-CAP 2003 Conference, pp. 121–128 (2003)
13. Sheth, A., Larson, J.: Federated database systems for managing distributed, heterogeneous, and autonomous databases. ACM Computing Surveys 22(3), 183–236 (1990)
14. Sheth, A., Kashyap, V.: So far (schematically) yet so near (semantically). IFIP Trans. A-25 41(5), 166–178 (1998)
15. Stuckenschmidt, H., Klein, M.: Modularization of ontologies, WonderWeb: Ontology infrastructure for the semantic web. Deliverable 21 v.0.6 (May 14, 2003)
16. Sahoo, S.S., Thomas, C., Sheth, A., York, W.S., Tartir, S.: Knowledge modeling and its application in life sciences: A tale of two ontologies. In: Proceedings of the K-CAP 2003 Conference (2006)
17. Thalheim, B.: Engineering database component ware. In: Draheim, D., Weber, G. (eds.) TEAA 2006. LNCS, vol. 4473, pp. 1–15. Springer, Heidelberg (2007)
18. The Gene Ontology Consortium. Gene ontology: tool for the unification of biology. Nature Genetics 25, 25–29 (2000)

19. Taylor, M.E., Matuszek, C., Klimt, B., Witbrock, M.: Autonomous classification of knowledge into an ontology. In: Proceedings of the Twentieth International FLAIRS Conference (2007)
20. Teorey, T.J., Wei, G., Bolton, D.L., Koenig, J.A.: ER model clustering as an aid for user communication and documentation in database design. Communications of the ACM 32(8), 975–987 (1989)
21. Volz, R., Oberle, D., Studer, R.: Views for light-weight web ontologies. In: Proceedings of the ACM Symposium on Applied Computing (SAC), 2003, pp. 1168–1173. ACM, New York (2003)

2

Formal Properties of Modularisation

Boris Konev[1], Carsten Lutz[2], Dirk Walther[1], and Frank Wolter[1]

[1] University of Liverpool, UK
{konev,dwalther,wolter}@liverpool.ac.uk
[2] University of Bremen, Germany
clu@informatik.uni-bremen.de

Summary. Modularity of ontologies is currently an active research field, and many different notions of a module have been proposed. In this paper, we review the fundamental principles of modularity and identify formal properties that a robust notion of modularity should satisfy. We explore these properties in detail in the contexts of description logic and classical predicate logic and put them into the perspective of well-known concepts from logic and modular software specification such as interpolation, forgetting and uniform interpolation. We also discuss reasoning problems related to modularity.

2.1 Introduction

The benefits of modular ontologies are manifold. In ontology design, modularity supports the structured and controlled development of large ontologies, enables ontology design by multiple, possibly distributed designers and allows the re-use of (parts of) already existing ontologies. In ontology deployment and usage, modularity can be exploited for right-sizing large ontologies (by selecting and using only the relevant part) and to speed up reasoning. Alas, making full use of modularity is hampered by the fact that there are many different definitions of what a module in an ontology actually is. In fact, it seems unlikely that there can be a unique and generally accepted such definition because the desirable properties of a module strongly depends on the intended application.

In this paper, our aim is to provide guidance for choosing the right notion of modularity. In particular, we give a survey of possible options and identify three formal properties that a notion of modularity may or may not enjoy and that can be used to evaluate the robustness of this notion in the context of a given application. We analyze whether the surveyed notions of modularity satisfy these robustness properties and provide further guidance by discussing the computational complexity of central decision problems associated with modularity.

To make different notions of modularity comparable to each other, we have to agree on some general framework for studying ontologies and modules. Generally speaking, a module is a part of a complex system that functions independently from this system. To define what a *module in an ontology* is, we thus have to specify what it means for such a module to function independently from the containing ontology. It is not a priori obvious how this can be done. We start with adopting the following, abstract view of an ontology: an ontology $\mathcal{O}$ can be regarded as a black box that provides answers to queries about some vocabulary S of interest. The form that such queries take is one of the main distinguishing factors between different applications. Important cases include the following.

Classification. Classifying an ontology means to compute the sub-/superclass relationships between all atomic classes in the ontology. For example, if S is a vocabulary for buildings and architecture, then the query $\mathsf{Church} \sqsubseteq \mathsf{Building}$ asks whether every church is a building.

Subsumption queries. In other applications, one is interested in computing subsumption between complex class expressions. For example, if S is a biological vocabulary, then the query $\mathsf{Father} \sqsubseteq \mathsf{Living\_being} \sqcap \exists \mathsf{has\_child}.\top$ asks whether every father is a living being having a child.

Instance data. A popular application of ontologies is their use for providing a background theory when querying instance data. In this case, one is interested in instance queries that are posed to a knowledge base, which consists of an ontology and an ABox that stores instances of classes and relations. Note that we do not consider the ABox to be part of the ontology. To represent this setup in terms of queries posed to the ontology, we consider queries that consist of an instance query *together* with an ABox. For example, if S is a geographical vocabulary, then a query might consist of the instance data

$$\mathcal{A} = \{\mathsf{Country}(\mathsf{France}), \mathsf{Country}(\mathsf{Columbia}), \ldots, \mathsf{LocatedinEurope}(\mathsf{France}), \ldots\}$$

together with the conjunctive query $\mathsf{EuropeanCountry}(\mathsf{France})$. This query asks whether it follows from the instance data $\mathcal{A}$ and the ontology that France is a European country. The answer is yes if, for example, the ontology states that every country $\mathsf{LocatedinEurope}$ is a $\mathsf{EuropeanCountry}$.

These examples show that, to define what it means for a part of an ontology to "function independently", we first have to fix a query language and a vocabulary of interest. Once this is done, two ontologies can be regarded equivalent if they give the same answers to all queries that can be built in the fixed query language with the fixed vocabulary. Similarly, given an ontology and its module, we can say that the module functions independently from the ontology if any query built in the fixed query language with the vocabulary associated with the module has the same answer when querying the module and the whole ontology.

Formally, these ideas can be captured by the concept of *inseparability*: given a query language $\mathcal{QL}$ and a vocabulary S, two ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$ are *S-inseparable w.r.t. $\mathcal{QL}$* if they give the same answers to queries in $\mathcal{QL}$ over S. There are various ways in which this notion can be used to define modularity of ontologies. For example, one can use the notion of inseparability to define a module as a subset of an ontology that is S-inseparable (and thus functions independently) from the whole ontology w.r.t. a query language $\mathcal{QL}$ and for a signature S associated with the module. In this case, investigations into modularity boil down to investigating conservative extensions as the special case of S-inseparability in which one ontology is a subset of the other ontology and the vocabulary S coincides with the vocabulary of the smaller ontology [13]. Another, differently flavoured approach is to use triples $(\mathcal{O}, \mathcal{QL}, S)$ as a module, where $\mathcal{O}$ is an ontology, $\mathcal{QL}$ a query language and S a vocabulary. The pair $(\mathcal{QL}, S)$ serves as an interface so that groups of independent modules can interact by querying each other via this interface. This setup is similar in spirit to formalisms such as DDLs and E-Connections [31, 17, 44, 4, 6]. In this case, S-inseparability is fundamental because it allows to define what it means that one module is equivalent to another one.

Since inseparability is at the core of most notions of modularity, our framework for studying and comparing such notions puts inseparability (and, as a variant, conservative extensions) into its focus. The robustness properties mentioned above are formulated directly in terms of inseparability. Here, the term "robustness" refers to the ramifications of changing the signature and manipulating in certain natural ways the involved ontologies. In particular, the properties ensure that modules and ontologies can be composed and decomposed in a transparent way. Our robustness properties are closely related to well-known notions from logic, in particular to interpolation and the Robinson joint consistency property. We explore this connection and also investigate the relation between inseparability on the one hand, and forgetting and uniform interpolation on the other. In principle, the general ideas presented in this paper are independent of the ontology language and query language that are used. To analyze robustness properties in concrete cases, though, we obviously have to fix both languages. As an ontology language, we mainly consider description logics (DLs), which are a highly relevant in this context due to the standardisation of the DL-based ontology language OWL by the W3C [5]. To round off our presentation, we will sometimes also give examples in terms of first- and second-order logic. We consider a wide range of query languages including classification queries, subsumption queries, instance queries, conjunctive queries and first- and second-order queries.

The rest of this paper is organised as follows. In Section 2.2, we introduce ontology languages and description logics. In Section 2.3, we define our framework for inseparability and introduce relevant robustness properties that notions of inseparability should enjoy. A variety of query languages together with the resulting robustness properties and decision problems are discussed in Section 2.5. A detailed investigation of robustness properties and their relation to interpolation is provided in Section 2.6. In Section 2.7 we explore the connection between inseparability and forgetting/uniform interpolation and establish a number of results regarding the existence of uniform interpolants and ontologies that "forget" certain symbols from a given ontology. Finally, we devote Section 2.8 to surveying results on lightweight DLs and acyclic ontologies. We finish with a brief discussion in Section 2.9.

2.2 Ontology Languages and Description Logics

We introduce a number of description logics, fix conventions for handling first- and second-order logic and give a number of basic definitions concerning signatures. The DL-literate reader may choose to skip this section.

2.2.1 The Description Logic $\mathcal{ALC}$

In DLs, ontologies are sets of implications between concepts, sometimes enriched with additional types of constraints. Thus, we start this section with introducing concepts, which are inductively defined from a set $\mathsf{N_C}$ of *concept names* and a set $\mathsf{N_R}$ of *role names*, using a set of *concept constructors*. From the perspective of first-order logic, concept names are unary predicates and role names are binary relations. Different sets of concept constructors give rise to different DLs.

In this paper, we will mainly be concerned with the description logic $\mathcal{ALC}$and its extensions. The concept constructors available in $\mathcal{ALC}$are shown in Table 2.1, where r denotes a role name and C and D denote concepts. A concept built from these constructors is called an *$\mathcal{ALC}$-concept*. A *concept implication* is an expression of the form $C \sqsubseteq D$, with C and D concepts. We write $C \equiv D$ instead of the two concept implications $C \sqsubseteq D$ and $D \sqsubseteq C$.

Definition 1 ($\mathcal{ALC}$-Ontology). An *$\mathcal{ALC}$-ontology* is a finite set of concept implications. $\mathcal{ALC}$-ontologies will also be called *$\mathcal{ALC}$-TBoxes*. ◇

Some of the applications discussed in this paper are concerned not only with ontologies, but also with instance data. In DLs, such instance data is described using ABoxes. To introduce ABoxes, we fix a set $\mathsf{N_I}$ of *individual names*, which correspond to constants in first-order logic. Then, an *assertion* is an expression of the form $C(a)$ or $r(a, b)$, where C is a concept, r a role name and a, b are individual names. An *$\mathcal{ALC}$-ABox* is a finite set of assertions. We call the combination $\mathcal{K} = (\mathcal{O}, \mathcal{A})$ of an $\mathcal{ALC}$-ontology and an $\mathcal{ALC}$-ABox an *$\mathcal{ALC}$-knowledge base*.

DL semantics is based on the notion of an *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$. The *domain* $\Delta^{\mathcal{I}}$ is a non-empty set and the *interpretation function* $\cdot^{\mathcal{I}}$ maps each concept

Table 2.1. Syntax and semantics of $\mathcal{ALC}$

Name	Syntax	Semantics
top concept	$\top$	$\Delta^{\mathcal{I}}$
bottom concept	$\bot$	$\emptyset$
negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
disjunction	$C \sqcup D$	$\neg(\neg C \sqcap \neg D)^{\mathcal{I}}$
existential restriction	$\exists r.C$	$\{d \in \Delta^{\mathcal{I}} \mid \exists e \in C^{\mathcal{I}} : (d, e) \in r^{\mathcal{I}}\}$
universal restriction	$\forall r.C$	$\neg(\exists r.\neg C)^{\mathcal{I}}$

name $A \in \mathsf{N_C}$ to a subset $A^\mathcal{I}$ of $\Delta^\mathcal{I}$, each role name $r \in \mathsf{N_R}$ to a binary relation $r^\mathcal{I}$ on $\Delta^\mathcal{I}$ and each individual name $a \in \mathsf{N_I}$ to an individual $a^\mathcal{I} \in \Delta^\mathcal{I}$. The extension of $\cdot^\mathcal{I}$ to arbitrary concepts is defined inductively as shown in the third column of Table 2.1.

The semantics of $\mathcal{ALC}$-ontologies is now defined as follows. An interpretation $\mathcal{I}$ *satisfies* a concept implication $C \sqsubseteq D$ if $C^\mathcal{I} \subseteq D^\mathcal{I}$, and $\mathcal{I}$ is a *model* of an ontology $\mathcal{O}$ if it satisfies all implications in $\mathcal{O}$. An ontology is *consistent* if it has a model. A concept C is *satisfiable w.r.t. an ontology* $\mathcal{O}$ if there exists a model $\mathcal{I}$ of $\mathcal{O}$ such that $C^\mathcal{I} \neq \emptyset$. A concept C is *subsumed by a concept D w.r.t. an ontology* $\mathcal{O}$ (written $\mathcal{O} \models C \sqsubseteq D$) if every model $\mathcal{I}$ of $\mathcal{O}$ satisfies the implication $C \sqsubseteq D$.

Now for the semantics of ABoxes and knowledge bases. An interpretation $\mathcal{I}$ *satisfies* an assertion $C(a)$ if $a^\mathcal{I} \in C^\mathcal{I}$, and $r(a,b)$ if $(a^\mathcal{I}, b^\mathcal{I}) \in r^\mathcal{I}$. It is a *model* of an ABox $\mathcal{A}$ if it satisfies all assertions in $\mathcal{A}$ and of a knowledge base $\mathcal{K} = (\mathcal{O}, \mathcal{A})$ if it is a model of both $\mathcal{O}$ and $\mathcal{A}$. We say that $\mathcal{A}$ is *consistent* if it has a model, and likewise for $\mathcal{K}$.

When working with ontologies and without ABoxes, the most relevant way of querying is *subsumption*: given an ontology $\mathcal{O}$ and concepts C, D, check whether $\mathcal{O} \models C \sqsubseteq D$. In the presence of ABoxes, there are two prominent ways of querying: instance checking and conjunctive query answering; see, e.g., [23, 10, 11]. *Instance checking* means, given a knowledge base $\mathcal{K}$ and an assertion $C(a)$, to check whether each model of $\mathcal{K}$ satisfies $C(a)$. If this is the case, we write $\mathcal{K} \models C(a)$.

Example 1. Let $\mathcal{O}$ be a geographical ontological defined as

$$\mathcal{O} = \{\mathsf{European\_Country} \equiv \mathsf{Country} \sqcap \mathsf{Located\_in\_Europe}\}$$

and $\mathcal{A}$ an ABox defined as

$$\mathcal{A} = \{\mathsf{Country(France)}, \mathsf{Country(Columbia)}, \ldots, \mathsf{Located\_in\_Europe(France)}, \ldots\}.$$

Then $(\mathcal{O}, \mathcal{A}) \models \mathsf{European\_Country(France)}$. $\diamond$

To discuss conjunctive query answering, we need a few preliminaries. An *atom* is of the form $C(v)$ or $r(v, v')$, where v, v' are from a *set of variables* $\mathsf{N_V}$, C is a concept and r is a role name. An *$\mathcal{ALC}$-conjunctive query* is an expression of the form $\exists \mathbf{v}.\varphi(\mathbf{u}, \mathbf{v})$, where $\mathbf{v}$ and $\mathbf{u}$ are disjoint sequences of variables and φ is a conjunction of atoms using only variables from $\mathbf{v} \cup \mathbf{u}$ (we confuse vectors and sets when convenient). The *arity* of such a query is the length of $\mathbf{u}$. The variables in $\mathbf{u}$ are the *answer variables* of q, and the ones in $\mathbf{v}$ are the (existentially) *quantified variables*. Let $\mathcal{K}$ be a knowledge base and $q = \exists \mathbf{v}.\varphi(\mathbf{u}, \mathbf{v})$ an n-ary conjunctive query. Then a sequence $\mathbf{a}$ of individual names of length n is a *certain answer* to $\mathcal{K}$ and q if for every model $\mathcal{I}$ of $\mathcal{K}$, there is a mapping $\pi : \mathbf{v} \cup \mathbf{u} \rightarrow \Delta^\mathcal{I}$ such that

- if v is the i-th element of $\mathbf{u}$ and a the i-th element of $\mathbf{a}$, then $\pi(v) = a^\mathcal{I}$;
- $C(v) \in \varphi$ implies $\pi(v) \in C^\mathcal{I}$, and $r(v, v') \in \varphi$ implies $(\pi(v), \pi(v')) \in r^\mathcal{I}$.

Then, *conjunctive query answering* means to compute, given a knowledge base $\mathcal{K}$ and conjunctive query q, all certain answers to $\mathcal{K}$ and q.

For our purposes, it usually suffices to work with an *instantiated conjunctive query* $\exists \mathbf{v}.\varphi(\mathbf{a}, \mathbf{v})$, in which the answer variables have been replaced with individual names. We write $\mathcal{K} \models \exists \mathbf{v}.\varphi(\mathbf{a}, \mathbf{v})$ if $\mathbf{a}$ is a certain answer to the query $\exists \mathbf{v}.\varphi(\mathbf{u}, \mathbf{v})$.

2.2.2 First- and Second-Order Logic

We use standard notation for first- and second-order logic. Throughout the paper, we admit individual constants, truth constants $\top$ and $\bot$, a binary equality symbol '$=$' and an arbitrary number of predicates of any arity. Function symbols are not admitted. A *first-order ontology*, or *$\mathcal{FO}$-ontology* for short, is simply a finite set of first-order sentences. As usual, we write $\mathcal{O} \models \varphi$ if a first-order sentence φ is a consequence of an $\mathcal{FO}$-ontology $\mathcal{O}$.

We will often view an $\mathcal{ALC}$-ontology as an $\mathcal{FO}$-ontology. Indeed, it is well-known that most DLs such as $\mathcal{ALC}$can be conceived as (decidable) fragments of first-order logic [3]. Note that a DL interpretation is just a first-order interpretation restricted to only unary and binary predicates and constants. Then, (i) concepts correspond to formulas in one free variable, (ii) concept implications and ABox assertions correspond to sentences and (iii) ontologies, ABoxes and knowledge bases correspond to first-order theories.

In what follows, we use $C^\sharp$ to denote the standard translation of an $\mathcal{ALC}$-concept C into an $\mathcal{FO}$-formula with one free variable x; see [3]. Thus, we have $A^\sharp = A(x)$ for every concept name A and, inductively,

$$\begin{aligned}
\top^\sharp &= x = x\\
\bot^\sharp &= \neg(x = x)\\
(C_1 \sqcap C_2)^\sharp &= C_1^\sharp \wedge C_2^\sharp\\
(\neg C)^\sharp &= \neg C^\sharp\\
(\exists r.C)^\sharp &= \exists y\, (r(x,y) \wedge C^\sharp[x/y])
\end{aligned}$$

where, in the last clause, y is a fresh variable. Then we can translate an ontology $\mathcal{O}$ into a corresponding $\mathcal{FO}$-ontology

$$\mathcal{O}^\sharp := \big\{\forall x\, \big(\, C^\sharp(x) \rightarrow D^\sharp(x)\big) \mid C \sqsubseteq D \in \mathcal{O}\big\}.$$

Thus, subsumption in $\mathcal{ALC}$can be understood in terms of $\mathcal{FO}$-consequence: for all $\mathcal{ALC}$-concepts C, D, we have $\mathcal{O} \models C \sqsubseteq D$ iff $\mathcal{O}^\sharp \models \forall x.(C^\sharp \rightarrow D^\sharp)$. We can translate a knowledge base $\mathcal{K}$ into an $\mathcal{FO}$-ontology $\mathcal{K}^\sharp$ in a similar way, using individual constants. Then, instance checking and checking whether a tuple $\mathbf{a}$ is a certain answer to a conjunctive query can also be understood as first order consequence.

Example 2. Let

$$\mathcal{O} = \{\mathsf{Father} \equiv \mathsf{Male} \sqcap \exists \mathsf{has\_child}.\top\}$$

define a father as a male who has a child. Then

$$\mathcal{O}^\sharp = \{\forall x\, (\mathsf{Father}(x) \leftrightarrow (\mathsf{Male}(x) \wedge \exists y\ \mathsf{has\_child}(x,y)))\}. \qquad \diamond$$

Second-order logic extends first-order logic by means of quantification over variables P for sets and relations. An $\mathcal{SO}$-ontology $\mathcal{O}$ is a finite set of $\mathcal{SO}$-sentences. We write $\mathcal{O} \models \varphi$ if a second-order sentence φ follows from an $\mathcal{SO}$-ontology $\mathcal{O}$. Clearly, every $\mathcal{FO}$-ontology is an $\mathcal{SO}$-ontology as well.

2.2.3 Signatures

The notion of a signature plays an important role in this paper. In a nutshell, a *signature* is a finite set of extra-logical symbols, i.e., symbols whose interpretation is not fixed a priori by the semantics. In the context of DLs, a signature may contain concept names, role names and individual names. Logical symbols such as the truth constants $\bot$, $\top$ and the Boolean operators $\sqcap$ and $\neg$ are not part of a signature. In the context of first- and second-order logic, a signature consists of predicate symbols (except equality) and individual constants. The equality symbol is not included because its interpretation is fixed a priori.

The *signature* $\mathsf{sig}(\mathcal{O})$ *of an $\mathcal{ALC}$-ontology* $\mathcal{O}$ is the set of concept and role names that occur in $\mathcal{O}$, and likewise for the signature $\mathsf{sig}(C)$ of a concept C and $\mathsf{sig}(C \sqsubseteq D)$ of a concept inclusion $C \sqsubseteq D$. For example,

$$\mathsf{sig}(\{\top \sqsubseteq \exists r.B \sqcap \forall r.\bot\}) = \{B, r\}.$$

The signature $\mathsf{sig}(\alpha)$ of an ABox assertion α, $\mathsf{sig}(\mathcal{A})$ of an ABox $\mathcal{A}$ and $\mathsf{sig}(\mathcal{K})$ of a knowledge base $\mathcal{K}$ is defined similarly, but additionally includes all occurring individual names.

The *signature* $\mathsf{sig}(\varphi)$ of a first- or second-order formula φ is the set of all predicate and constant symbols (except equality) used in φ. Note that $\mathsf{sig}(\forall P.\varphi) = \mathsf{sig}(\exists P.\varphi) = \mathsf{sig}(\varphi)$ for every $\mathcal{SO}$-formula φ. This notion is lifted to ontologies in the obvious way. For an $\mathcal{SO}$-sentence φ and a relation symbol S we sometimes write $\exists S.\varphi$ instead of $\exists P.\varphi[P/S]$, where P is a variable for relations of the same arity as S and $\varphi[P/S]$ results from φ by replacing S by P. Clearly, $\mathsf{sig}(\exists S.\varphi) = \mathsf{sig}(\varphi) \setminus \{S\}$.

In this paper, we are often interested in concepts and ontologies that are formulated using a specific signature. Therefore, we talk of an S-ontology $\mathcal{O}$ if $\mathsf{sig}(\mathcal{O}) \subseteq S$, and likewise for S-concepts, etc. When we want to emphasise both the DL $\mathcal{L}$ in which an ontology is formulated and the signature S, we talk about $\mathcal{L}_S$-ontologies.

2.2.4 Some Extensions of $\mathcal{ALC}$

We introduce here the most important extensions of $\mathcal{ALC}$used in this paper. Some additional extensions (and fragments) are introduced as needed. The extensions considered here fall into three categories: (i) additional concept constructors, (ii) additional roles and (iii) additional statements in TBoxes. The extensions are listed in Table 2.2, where $\#X$ denotes the size of a set X and double horizontal lines mark the border between extensions of type (i), (ii) and (iii), from top to bottom. The last column lists the identifier for each extension, which is simply appended to the name

Table 2.2. Additional constructors: syntax and semantics

Name	Syntax	Semantics	Identifier
number restrictions	$(\leqslant n\ r\ C)$ $(\geqslant n\ r\ C)$	$\{d \mid \#\{e \mid (d,e) \in r^{\mathcal{I}} \wedge e \in C^{\mathcal{I}}\} \leq n\}$ $\{d \mid \#\{e \mid (d,e) \in r^{\mathcal{I}} \wedge e \in C^{\mathcal{I}}\} \geq n\}$	$\mathcal{Q}$
nominals	$\{a\}$	$\{a^{\mathcal{I}}\}$	$\mathcal{O}$
inverse role	r^-	$(r^{\mathcal{I}})^{-1}$	$\mathcal{I}$
universal role	u	$\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$	$\mathcal{U}$
role inclusions	$r \sqsubseteq s$	$r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$	$\mathcal{H}$

$\mathcal{ALC}$. For example, $\mathcal{ALC}$extended with number restrictions and inverse roles is denoted by $\mathcal{ALCQI}$ and the extension of a DL $\mathcal{L}$ with the universal role is denoted by $\mathcal{LU}$.

In the following, we only give some remarks regarding the listed extensions and refer the reader to the DL literature for more details [3]. When inverse roles are present, they can be used inside existential and universal restrictions, and also inside number restrictions (if present) and role inclusions (if present). In contrast, the universal role u is only allowed inside existential and universal restrictions. We remark that the universal role is less common than the other extensions of $\mathcal{ALC}$, but it will play a prominent role in this paper.

Some care is required for defining the signature in extensions of $\mathcal{ALC}$. If we work with a description logic $\mathcal{L}$ that includes nominals, then signatures (of concepts, ontologies, ABoxes, etc.) include also the individual names a occurring as a nominal $\{a\}$. For example, the signature of the $\mathcal{ALCO}$-ABox $\{(A \sqcap \exists r.\{a\})(b)\}$ is $\{A, r, a, b\}$. In contrast to nominals, the universal role is regarded as a logical symbol and is not part of the signature. This is justified by the fact that the interpretation of the universal role u is fixed a priori. Note also that for its translation $(\exists u.D)^\sharp = \exists x.D^\sharp$ to first-order logic, no relation symbol is required.

2.3 Inseparability and Conservative Extensions

We lay the foundation of the general framework that we use to study modularity. This framework consists of conventions that fix (in a rather liberal way) what an ontology, an ontology language, a query, and a query language is, and of the fundamental notions that we use to define modularity: S-inseparability and S-conservative extensions. We also introduce the relevant decision problems associated with S-inseparability and S-conservative extensions.

2.3.1 Basic Notions and Conventions

For us, an *ontology* is a finite set of second-order sentences, an *ontology language* is a (commonly infinite) set of second-order sentences, a *query* is a second-order sentence, and a *query language* is a (finite or infinite) set of queries. By the first-order translations given in Section 2.2, this view captures ontologies and query languages based on DLs. Observe that ontology languages and query languages are defined in the same way, and we will often make use of that. We now define the core notions for defining modularity.

Definition 2 (Inseparability and Conservative Extension). Let $\mathcal{QL}$ be a query language, $\mathcal{O}_1, \mathcal{O}_2$ ontologies and S a signature. We say that

- $\mathcal{O}_1$ and $\mathcal{O}_2$ are *S-inseparable* w.r.t. $\mathcal{QL}$ and write $\mathcal{O}_1 \approx_S^{\mathcal{QL}} \mathcal{O}_2$ iff for all $\varphi \in \mathcal{QL}$ with $\mathsf{sig}(\varphi) \subseteq S$, we have $\mathcal{O}_1 \models \varphi$ iff $\mathcal{O}_2 \models \varphi$.
- $\mathcal{O}_2$ is an *S-conservative extension of* $\mathcal{O}_1$ w.r.t. $\mathcal{QL}$ iff $\mathcal{O}_2 \supseteq \mathcal{O}_1$ and $\mathcal{O}_1$ and $\mathcal{O}_2$ are S-inseparable w.r.t. $\mathcal{QL}$. If, in addition, $S = \mathsf{sig}(\mathcal{O}_1)$, then we say that $\mathcal{O}_2$ is a *conservative extension of* $\mathcal{O}_1$ w.r.t. $\mathcal{QL}$.

We say that $\varphi \in \mathcal{QL}$ *separates* $\mathcal{O}_1$ *and* $\mathcal{O}_2$ iff $\mathcal{O}_1 \models \varphi$ and $\mathcal{O}_2 \not\models \varphi$ or vice versa. ◇

Observe that if $\mathcal{O}_1 \subseteq \mathcal{O}_2$ are formulated in first-order logic and $\mathcal{QL}$ consists of all first-order sentences, then our definition of a conservative extension w.r.t. $\mathcal{QL}$ coincides with the standard definition of a conservative extension used in mathematical logic [13].

For any query language $\mathcal{QL}$ and any signature S, the relation $\approx_S^{\mathcal{QL}}$ of S-inseparability w.r.t. $\mathcal{QL}$ is clearly an equivalence relation. Moreover, the following two implications are easily seen to hold:

1. if $\mathcal{O}_1 \approx_S^{\mathcal{QL}} \mathcal{O}_2$ and $S' \subseteq S$, then $\mathcal{O}_1 \approx_{S'}^{\mathcal{QL}} \mathcal{O}_2$;
2. if $\mathcal{O}_1 \approx_S^{\mathcal{QL}} \mathcal{O}_2$ and $\mathcal{QL}' \subseteq \mathcal{QL}$, then $\mathcal{O}_1 \approx_S^{\mathcal{QL}'} \mathcal{O}_2$.

The largest query language $\mathcal{QL}$ considered in this paper is $\mathcal{QL}_{\mathcal{SO}}$, the set of sentences of second-order logic. It follows from Point 2 above that inseparability w.r.t. $\mathcal{QL}_{\mathcal{SO}}$ implies inseparability w.r.t. any other query language that fits into our framework.

2.3.2 Examples

We give several examples that illustrate the importance of S-inseparability and conservative extensions for handling ontologies in general and for defining notions of modularity in particular. For simplicity, we concentrate on the rather expressive query language $\mathcal{QL}_{\mathcal{SO}}$ for now and defer the introduction of DL-based query languages to the subsequent section.

Example 3. The following ontology defines the binary relation P ('part of') as a transitive and reflexive relation:

$$\begin{aligned}\mathcal{O}_1 = \{ & \forall x \mathsf{P}(x,x),\\ & \forall x \forall y \forall z\ (\mathsf{P}(x,y) \wedge \mathsf{P}(y,z) \rightarrow \mathsf{P}(x,z)) \}\end{aligned}$$

We now define another ontology that also defines P, but which does this in terms of another binary relation PP ('proper part of') that is transitive and irreflexive:

$$\begin{aligned}\mathcal{O}_2 = \{ & \forall x\, \neg\mathsf{PP}(x,x),\\ & \forall x\forall y\forall z\ (\mathsf{PP}(x,y)\wedge\mathsf{PP}(y,z)\rightarrow\mathsf{PP}(x,z)),\\ & \forall x\forall y\ (\mathsf{P}(x,y)\leftrightarrow(x=y\vee\mathsf{PP}(x,y))\ \}.\end{aligned}$$

Suppose that the ontology $\mathcal{O}_2$ is used in an application that refers only the predicate P (but not PP) and is based on the query language $\mathcal{QL}$. Can we replace $\mathcal{O}_2$ with the somewhat simpler ontology $\mathcal{O}_1$ without causing any changes or even corruption? In our framework, answering this question means checking whether or not $\mathcal{O}_1$ and $\mathcal{O}_2$ are $\{\mathsf{P}\}$-inseparable w.r.t. $\mathcal{QL}_{SO}$. The answer is 'no': the $\mathcal{QL}_{SO}$-sentence

$$\varphi = \forall x\forall y\ ((\mathsf{P}(x,y)\wedge\mathsf{P}(y,x))\rightarrow x=y)$$

separates $\mathcal{O}_1$ and $\mathcal{O}_2$ since $\mathcal{O}_1 \not\models \varphi$ and $\mathcal{O}_2 \models \varphi$. While $\mathcal{O}_1$ and $\mathcal{O}_2$ are toy ontologies, questions of this kind are of obvious importance for real applications. To conclude this example, we remark that $\mathcal{O}_1$ and $\mathcal{O}_2$ become $\{\mathsf{P}\}$-inseparable w.r.t. $\mathcal{QL}_{SO}$ by adding φ to $\mathcal{O}_1$. ◇

Example 4. According to the introduction, a module should function independently from the containing ontology and conservative extensions can capture this. Thus, let us use conservative extensions to give an example definition of a module.

Definition 3. *Let $\mathcal{O}_1 \subseteq \mathcal{O}_2$ be ontologies, S a signature, and $\mathcal{QL}$ a query language. Then $\mathcal{O}_1$ is a* weak S-module *of $\mathcal{O}_2$ w.r.t. $\mathcal{QL}$ if $\mathcal{O}_2$ is an S-conservative extension of $\mathcal{O}_1$ w.r.t. $\mathcal{QL}$.* ◇

As a concrete example, consider the $\mathcal{ALC}$-ontology

$$\begin{aligned}\mathcal{O}_1 = \{ \quad & \mathsf{Male} \equiv \mathsf{Human} \sqcap \neg\mathsf{Female},\\ & \mathsf{Human} \sqsubseteq \forall \mathsf{has\_child}.\mathsf{Human}\ \}\end{aligned}$$

and its extension

$$\mathcal{O}_2 = \mathcal{O}_1 \cup \{\mathsf{Father} \equiv \mathsf{Male} \sqcap \exists\mathsf{has\_child}.\top\}.$$

If $S = \mathsf{sig}(\mathcal{O}_1)$, then $\mathcal{O}_1$ is a weak S-module of $\mathcal{O}_2$. ◇

Example 5. Consider again the ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$ from Example 4. The extension of $\mathcal{O}_1$ to $\mathcal{O}_2$ is simple, but a rather typical case in practice: one or more newly introduced concept names are defined in terms of concept and role names that are already defined in the existing ontology. The existing concept and role names are only *used*, but not affected in any way by this extension. In particular, no new consequences referring only to previously existing concept and role names is derivable, and thus the extension is conservative. Extensions of this simple kind are called *definitorial.*

The notion of a module defined in Definition 3 is much more liberal than this: for example, it allows the extension to make statements about existing symbols as long as these statements are already entailed by the existing ontology. As another example for the definition of a module, we give a stronger one that is closer to the idea of definitorial extensions:

Definition 4. *Let $\mathcal{O}_1 \subseteq \mathcal{O}_2$ be ontologies, S a signature and $\mathcal{QL}$ a query language. Then $\mathcal{O}_1$ is a* strong S-module *of $\mathcal{O}_2$ w.r.t. $\mathcal{QL}$ if it is a weak S-module of $\mathcal{O}_2$ w.r.t. $\mathcal{QL}$ and, in addition, $\mathcal{O}_2 \setminus \mathcal{O}_1$ and the empty ontology are S-inseparable w.r.t. $\mathcal{QL}$.* $\diamond$

To see an example that illustrates the difference between Definitions 3 and 4, let

$$\mathcal{O}_2' = \mathcal{O}_1 \cup \{\mathsf{Father} \sqsubseteq \mathsf{Human}, \mathsf{Father} \equiv \mathsf{Male} \sqcap \exists \mathsf{has\_child}.\top\}.$$

Let $S = \mathsf{sig}(\mathcal{O}_1)$. Then $\mathcal{O}_1$ is not a strong S-module of $\mathcal{O}_2'$ because

$$\mathcal{O}_2' \setminus \mathcal{O}_1 \models \mathsf{Male} \sqcap \exists \mathsf{has\_child}.\top \sqsubseteq \mathsf{Human}$$

and this inclusion does not follow from the empty ontology. However, we already have $\mathcal{O}_1 \models \mathsf{Male} \sqcap \exists \mathsf{has\_child}.\top \sqsubseteq \mathsf{Human}$ and, indeed, it is possible to prove that $\mathcal{O}_1$ is a weak S-module of $\mathcal{O}_2'$. $\diamond$

We do not claim that Definitions 3 and 4 are the only reasonable definitions of a module. As indicated already in the introduction, many subtle variations and even completely different approaches are possible. What is common to virtually all definitions of a module is that inseparability and conservative extensions play a central role. In this paper, we will not favour a particular definition of a module, but rather study modularity directly in terms of inseparability and conservative extensions. We also remark that there are other interesting applications of inseparability such as ontology versioning and module extraction [30, 28, 27, 14].

2.3.3 Decision Problems

The notions of inseparability and conservative extension give rise to decision problems in a natural way. Let $\mathcal{L}$ be an ontology language and $\mathcal{QL}$ a query language. Then

- the *S-inseparability problem for* $(\mathcal{L}, \mathcal{QL})$ is to decide, given $\mathcal{L}$-ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$ and a signature S, whether $\mathcal{O}_1 \approx_S^{\mathcal{QL}} \mathcal{O}_2$;
- the *S-conservativity problem for* $(\mathcal{L}, \mathcal{QL})$ is to decide, given $\mathcal{L}$-ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$ with $\mathcal{O}_1 \subseteq \mathcal{O}_2$ and a signature S, whether $\mathcal{O}_1 \approx_S^{\mathcal{QL}} \mathcal{O}_2$;
- the *conservativity problem for* $(\mathcal{L}, \mathcal{QL})$ is to decide, given $\mathcal{L}$-ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$ with $\mathcal{O}_1 \subseteq \mathcal{O}_2$ whether $\mathcal{O}_1 \approx_S^{\mathcal{QL}} \mathcal{O}_2$ with $S = \mathsf{sig}(\mathcal{O}_1)$;

In view of the examples given in this section, the utility of these problems should be clear: they can be used to decide whether a given part of an ontology is a module, whether two ontologies can be exchanged in some application, whether an extension to an ontology that is intended to be definitiorial has damaged the terms that were already defined, etc.

We will discuss these problems in more detail when surveying the options for query languages in Section 2.5. For now, we only remark that, in the special case where $\mathcal{L} \subseteq \mathcal{QL}$ and $S \supseteq \mathsf{sig}(\mathcal{O}_1 \cup \mathcal{O}_2)$, it is not hard to see that

$$\mathcal{O}_1 \approx_S^{\mathcal{QL}} \mathcal{O}_2 \text{ iff } \mathcal{O}_1 \models \varphi \text{ for all } \varphi \in \mathcal{O}_2 \text{ and } \mathcal{O}_2 \models \varphi \text{ for all } \varphi \in \mathcal{O}_1.$$

It follows that in this particular case checking S-inseparability reduces to logical entailment and thus to standard reasoning in $\mathcal{QL}$.

2.4 Robustness Properties

We introduce the three robustness properties for modularity. Since we study modularity in terms of inseparability, we also formulate these properties in terms of inseparability.

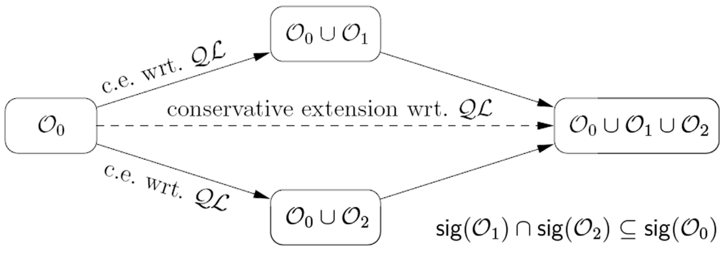
Definition 5 (Robustness Properties). Let $\mathcal{L}$ be an ontology language and $\mathcal{QL}$ a query language. Then $(\mathcal{L}, \mathcal{QL})$ is *robust*

- *under vocabulary extensions* if, for all $\mathcal{L}$-ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$ and signatures S, S' with $S' \cap \mathsf{sig}(\mathcal{O}_1 \cup \mathcal{O}_2) \subseteq S$, the following holds:
$$\mathcal{O}_1 \approx_S^{\mathcal{QL}} \mathcal{O}_2 \quad \Rightarrow \quad \mathcal{O}_1 \approx_{S'}^{\mathcal{QL}} \mathcal{O}_2;$$
- *under joins* if, for all $\mathcal{L}$-ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$ and signatures S with $\mathsf{sig}(\mathcal{O}_1) \cap \mathsf{sig}(\mathcal{O}_2) \subseteq S$, the following holds for $i = 1, 2$:
$$\mathcal{O}_1 \approx_S^{\mathcal{QL}} \mathcal{O}_2 \quad \Rightarrow \quad \mathcal{O}_i \approx_S^{\mathcal{QL}} \mathcal{O}_1 \cup \mathcal{O}_2;$$
- *under replacement* if, for all $\mathcal{L}$-ontologies $\mathcal{O}_1$, $\mathcal{O}_2$ and $\mathcal{O}$ and signatures S with $\mathsf{sig}(\mathcal{O}) \cap \mathsf{sig}(\mathcal{O}_1 \cup \mathcal{O}_2) \subseteq S$, the following holds:
$$\mathcal{O}_1 \approx_S^{\mathcal{QL}} \mathcal{O}_2 \quad \Rightarrow \quad \mathcal{O}_1 \cup \mathcal{O} \approx_S^{\mathcal{QL}} \mathcal{O}_2 \cup \mathcal{O}. \qquad \diamond$$

In the remainder of this section, we give examples that motivate the usefulness of the properties given in Definition 5.

Robustness under vocabulary extensions. In practice, most ontologies constantly evolve: they are regularly being extended, updated, corrected, etc. This evolution usually results in frequent modifications of the ontology's signature. For this reason, inseparability should be robust under changes of the signature, and this is what robustness under vocabulary extensions is about. In many applications, being inseparable or a conservative extension is of doubtful relevance when a simple addition of fresh symbols to the signature (i.e., symbols that are not mentioned in any of the involved ontologies at all) can change this situation.

Robustness under joins. Taking the union of two or more ontologies is an essential and frequently used operation. For example, the extension of an existing ontology



$\mathcal{O}_0$
$\mathcal{O}_0 \cup \mathcal{O}_1$
$\mathcal{O}_0 \cup \mathcal{O}_2$
$\mathcal{O}_0 \cup \mathcal{O}_1 \cup \mathcal{O}_2$
c.e. wrt. $\mathcal{QL}$
conservative extension wrt. $\mathcal{QL}$
c.e. wrt. $\mathcal{QL}$
$\mathsf{sig}(\mathcal{O}_1) \cap \mathsf{sig}(\mathcal{O}_2) \subseteq \mathsf{sig}(\mathcal{O}_0)$

Fig. 2.1. Robustness under joins

with new concept inclusions can be conceived as a union, and also the import of an existing ontology into a newly constructed one. Robustness under joins is concerned with the behaviour of inseparability regarding this operation. Its virtue is probably best appreciated by considering the following consequence of robustness under joins, formulated in terms of conservative extensions:

($*$) for all $\mathcal{L}$-ontologies $\mathcal{O}_0$, $\mathcal{O}_1$, $\mathcal{O}_2$, if $\mathcal{O}_0 \cup \mathcal{O}_1$ and $\mathcal{O}_0 \cup \mathcal{O}_2$ are conservative extensions of $\mathcal{O}_0$ w.r.t. $\mathcal{QL}$ and $\mathsf{sig}(\mathcal{O}_1) \cap \mathsf{sig}(\mathcal{O}_2) \subseteq \mathsf{sig}(\mathcal{O}_0)$, then $\mathcal{O}_0 \cup \mathcal{O}_1 \cup \mathcal{O}_2$ is a conservative extension of $\mathcal{O}_0$ w.r.t. $\mathcal{QL}$.

To see why ($*$) is useful, suppose that two (or more) ontology designers simultaneously extend an ontology $\mathcal{O}_0$, but work on different parts. They produce new sets of concept inclusions $\mathcal{O}_1$ and $\mathcal{O}_2$ to be added to $\mathcal{O}_0$ and ensure that $\mathcal{O}_0 \cup \mathcal{O}_i$ is a conservative extension of $\mathcal{O}_0$ w.r.t. the query language $\mathcal{QL}$ that is used in the intended application, for $i \in \{1, 2\}$. Now, robustness under joins guarantees (via ($*$)) that the joint extension $\mathcal{O}_0 \cup \mathcal{O}_1 \cup \mathcal{O}_2$ is also a conservative extension of $\mathcal{O}_0$ w.r.t. $\mathcal{QL}$, as illustrated in Figure 2.1. In the sketched situation, robustness under joins thus ensures that the two designers can work truly independently, i.e., they will not be forced to redesign $\mathcal{O}_1$ and $\mathcal{O}_2$ because of unexpected interactions.

Robustness under replacement. This property is critical when a module in an ontology is supposed to be *re-used* in another ontology. We consider two example scenarios.

Assume that an ontology designer develops an ontology $\mathcal{O}_0$ for hospital administration and wants to reuse a set S of medical terms from a medical ontology $\mathcal{O}_1$. A typical example for $\mathcal{O}_1$ would be SNOMED CT, the Systematized Nomenclature of Medicine, Clinical Terms [45]. The designer knows that queries to his ontology will be formulated in a query language $\mathcal{QL}$. Instead of importing the whole ontology $\mathcal{O}_1$, which potentially is very large (SNOMED CT comprises ∼0.4 million medical terms), he chooses to import a weak S-module $\mathcal{M}$ of $\mathcal{O}_1$ w.r.t. $\mathcal{QL}$, i.e., $\mathcal{M} \subseteq \mathcal{O}_1$ and $\mathcal{M} \approx_S^{\mathcal{QL}} \mathcal{O}_1$. If $(\mathcal{L}, \mathcal{QL})$ is robust under replacement, where $\mathcal{L}$ is the language in which $\mathcal{O}_0$ and $\mathcal{O}_1$ are formulated, then it follows that

$$\mathcal{M} \cup \mathcal{O}_0 \approx_S^{\mathcal{QL}} \mathcal{O}_1 \cup \mathcal{O}_0.$$

$$\begin{tikzcd}
\boxed{\mathcal{M}} \arrow[r, phantom, "\approx_S^{\mathcal{QL}}"] \arrow[d] & \boxed{\mathcal{O}_1} \arrow[d] \\
\boxed{\mathcal{M} \cup \mathcal{O}_0} \arrow[r, phantom, "\approx_S^{\mathcal{QL}}"] & \boxed{\mathcal{O}_1 \cup \mathcal{O}_0}
\end{tikzcd}$$

$$\mathsf{sig}(\mathcal{O}_0) \cap \mathsf{sig}(\mathcal{M} \cup \mathcal{O}_1) \subseteq S$$

Fig. 2.2. Robustness under replacement

It is thus indeed possible to use $\mathcal{M}$ instead of $\mathcal{O}_1$ in $\mathcal{O}_0$ without loosing consequences for the signature S in $\mathcal{QL}$; observe that this does not follow from the definition of a weak module alone. This situation is illustrated in Figure 2.2. Observe that the argument does not depend on the details of $\mathcal{O}_0$ and does not break when $\mathcal{O}_0$ evolves.

For a second example, we consider the same scenario. Assume now that the designer of the ontology $\mathcal{O}_0$ for hospital administration imports the whole ontology $\mathcal{O}_1$ to reuse the set S of medical terms. He designs $\mathcal{O}_0$ such that the terms from S as defined in $\mathcal{O}_1$ are not corrupted, formalised by $\mathcal{O}_0 \cup \mathcal{O}_1 \approx_S^{\mathcal{QL}} \mathcal{O}_1$. Also assume that the ontology $\mathcal{O}_1$ is updated frequently. The designer wants to ensure that, when an old version of $\mathcal{O}_1$ is replaced with a new one, the terms from S as given in the new version of $\mathcal{O}_1$ are still not corrupted. Since he cannot foresee the changes to $\mathcal{O}_1$ that will occur in the future, he needs to design $\mathcal{O}_0$ such that $\mathcal{O}_0 \cup \mathcal{O}_1 \approx_S^{\mathcal{QL}} \mathcal{O}_1$ for all ontologies $\mathcal{O}_1$ with $\mathsf{sig}(\mathcal{O}_0) \cap \mathsf{sig}(\mathcal{O}_1) \subseteq S$. If $(\mathcal{L}, \mathcal{QL})$ is robust under replacement, this is easy: it simply suffices to ensure that $\mathcal{O}_0 \approx_{\mathcal{L}}^{\mathcal{QL}} \emptyset$. This use case has been discussed in more detail in [16].

The importance of robustness under replacement has first been observed in [15, 14], see also [16]. It also plays an important role in the context of the inseparability of programs in logic programming and answer set programming [34, 19]. Finally, we make a note on the interplay between robustness under replacements and strong modules as introduced in Definition 4: if $(\mathcal{L}, \mathcal{QL})$ enjoys robustness under replacements and in the realistic case that $\mathsf{sig}(\mathcal{O}_2 \setminus \mathcal{O}_1) \cap \mathsf{sig}(\mathcal{O}_1) \subseteq S$, we have that $\mathcal{O}_1$ is a strong S-module of $\mathcal{O}_2$ w.r.t. $\mathcal{QL}$ iff $\mathcal{O}_2 \setminus \mathcal{O}_1$ and the empty ontology are S-inseparable w.r.t. $\mathcal{QL}$, i.e., the condition that $\mathcal{O}_1$ is a weak module of $\mathcal{O}_2$ is redundant.

2.5 Query Languages, Robustness and Complexity

So far, we have used the powerful query language $\mathcal{QL}_{\mathcal{SO}}$. For many applications, this is inappropriate because $\mathcal{QL}_{\mathcal{SO}}$ can separate ontologies for which no differences are observable in the application. In this section, we introduce a number of DL-based query languages used in applications and discuss the corresponding notions of S-inseparability and S-conservative extensions, as well as their robustness properties and the complexity of the decision problems given in Section 2.3.3. We start with rather weak query languages and gradually move towards more expressive ones including $\mathcal{QL}_{\mathcal{FO}}$ and $\mathcal{QL}_{\mathcal{SO}}$.

2.5.1 Inconsistency

A very weak query language $\mathcal{QL}_\bot$ can be defined as $\{\top \sqsubseteq \bot\}$, i.e., $\mathcal{QL}_\bot$ consists of an unsatisfiable first-order sentence. Clearly, $\mathcal{O} \models \top \sqsubseteq \bot$ iff $\mathcal{O}$ is inconsistent. Thus, two ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$ are S-inseparable w.r.t. $\mathcal{QL}_\bot$ if either both $\mathcal{O}_1$ and $\mathcal{O}_2$ are inconsistent, or both are consistent. Observe that S-inseparability w.r.t. $\mathcal{QL}_\bot$ does not depend on the actual signature S.

The query language $\mathcal{QL}_\bot$ is too weak for defining a reasonable notion of modularity. However, $\mathcal{QL}_\bot$ can be used to ensure that the modification of an ontology has not caused inconsistency: the extension $\mathcal{O}_2$ of an ontology $\mathcal{O}_1$ by a set of sentences $\mathcal{O}$ does not cause inconsistency iff $\mathcal{O}_2$ is a conservative extension of $\mathcal{O}_1$ w.r.t. $\mathcal{QL}_\bot$.

For every ontology language $\mathcal{L}$, $(\mathcal{L}, \mathcal{QL}_\bot)$ is robust under vocabulary extensions because S-inseparability does not depend on S. It is robust neither under joins nor replacement, for any of the ontology languages introduced in Section 2.2. Let, for example, $\mathcal{O}_1 = \{A \equiv \top\}$ and $\mathcal{O}_2 = \{A \equiv \bot\}$. Then $\mathcal{O}_1 \approx^{\mathcal{QL}_\bot} \mathcal{O}_2$, but $\mathcal{O}_1 \not\approx^{\mathcal{QL}_\bot} \mathcal{O}_1 \cup \mathcal{O}_2$ because $\mathcal{O}_1$ and $\mathcal{O}_2$ are consistent, but $\mathcal{O}_1 \cup \mathcal{O}_2$ is inconsistent. Thus, $(\mathcal{ALC}, \mathcal{QL}_\bot)$ is not robust under joins. The decision problems associated with $(\mathcal{L}, \mathcal{QL}_\bot)$ obviously have the same complexity as deciding ontology consistency in $\mathcal{L}$.

2.5.2 Subsumption between Concept Names

The query language $\mathcal{QL}_{\mathsf{CN}}$ is defined as the set of all queries $A \sqsubseteq B$, where A and B are concept names or truth constants $\bot$ and $\top$. Two ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$ are S-inseparable w.r.t. $\mathcal{QL}_{\mathsf{CN}}$ iff they give raise to the same classification regarding the concept names in S, i.e., the sub-/superclass relation between any two concept names in S is identical.

Using $\mathcal{QL}_{\mathsf{CN}}$ as a query language is useful if the application relies only on the classification, i.e., changes in the meaning of a symbol are considered acceptable as long as the classification does not change. Observe that S-inseparability w.r.t. $\mathcal{QL}_{\mathsf{CN}}$ implies S-inseparability w.r.t. $\mathcal{QL}_\bot$, but the converse does not hold whenever $S \neq \emptyset$. Also observe that S-inseparability w.r.t. $\mathcal{QL}_{\mathsf{CN}}$ is oblivious to the addition and deletion of role and individual names to and from S.

As illustrated by the following example, the query language $\mathcal{QL}_{\mathsf{CN}}$ is still very weak when used to define modularity.

Example 6. Reconsider the ontologies from Example 5:

$$\mathcal{O}_1 = \{\mathsf{Male} \equiv \mathsf{Human} \sqcap \neg\mathsf{Female}, \mathsf{Human} \sqsubseteq \forall \mathsf{has\_child}.\mathsf{Human}\}$$
$$\mathcal{O}_2' = \mathcal{O}_1 \cup \{\mathsf{Father} \sqsubseteq \mathsf{Human}, \mathsf{Father} \equiv \mathsf{Male} \sqcap \exists \mathsf{has\_child}.\top\}.$$

Let $S = \mathsf{sig}(\mathcal{O}_1)$. Then $\mathcal{O}_1$ is a weak S-module because the same implications between concept names in S are derivable from $\mathcal{O}_1$ and $\mathcal{O}_2'$. However, this is true only because we restrict ourselves to subsumption between concept names, as we had said already that $\mathsf{Male} \sqcap \exists\mathsf{has\_child}.\top \sqsubseteq \mathsf{Human}$ separates the two ontologies. ◇

It is easy to see that $(\mathcal{L}, \mathcal{QL}_{\mathsf{CN}})$ is robust under vocabulary extensions for any ontology language $\mathcal{L}$ introduced in Section 2.2. As shown by the next example, robustness under joins and replacement fails.

Example 7. Let $\mathcal{O}_1 = \{A \sqsubseteq \exists r.B\}$, $\mathcal{O}_2 = \{\exists r.B \sqsubseteq E\}$, and $S = \{A, B, r, E\}$. Then $\mathcal{O}_1 \approx_S^{\mathcal{QL}_{\mathsf{CN}}} \mathcal{O}_2$. Failure of robustness under joins follows from $\mathcal{O}_1 \not\approx_S^{\mathcal{QL}_{\mathsf{CN}}} \mathcal{O}_1 \cup \mathcal{O}_2$ since $\mathcal{O}_1 \cup \mathcal{O}_2 \models A \sqsubseteq E$ and $\mathcal{O}_1 \not\models A \sqsubseteq E$.

For failure of robustness under replacement consider $\mathcal{O} = \{A \sqsubseteq \neg\exists r.B\}$. Then $\mathcal{O}_1 \cup \mathcal{O} \models A \sqsubseteq \bot$ but $\mathcal{O}_2 \cup \mathcal{O} \not\models A \sqsubseteq \bot$. ◇

Still, $\mathcal{QL}_{\mathsf{CN}}$ has useful applications, e.g., for efficient implementations of classification in a DL reasoner and in distributed description logics, where subsumptions between concept names are usually all that matter. The decision problems associated with $(\mathcal{L}, \mathcal{QL}_{\mathsf{CN}})$ are not harder than deciding subsumption between concept names in $\mathcal{L}$.

2.5.3 Subsumption between Complex Concepts

For any description logic $\mathcal{L}$, the query language $\mathcal{QL}_{\mathcal{L}}$ is defined as the set of all queries $C \sqsubseteq D$, where C and D are $\mathcal{L}$-concepts. If $\mathcal{L}$ admits role inclusions, we also include queries $r \sqsubseteq s$, with r and s roles. Thus, two ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$ are S-inseparable w.r.t. $\mathcal{QL}_{\mathcal{L}}$ iff they entail the same subsumptions between $\mathcal{L}_S$-concepts and the same role inclusions $r \sqsubseteq s$ with $r, s \in S$, if role inclusions are present in $\mathcal{L}$. The following example shows that even for propositional DLs $\mathcal{L}$ without role names, $\mathcal{QL}_{\mathcal{L}}$ is strictly stronger than $\mathcal{QL}_{\mathsf{CN}}$.

Example 8. Let

$$\mathcal{O}_1 = \emptyset, \quad \mathcal{O}_2 = \{\mathsf{Parent} \sqcap \mathsf{Male} \sqsubseteq \mathsf{Father}\}, \quad S = \{\mathsf{Parent}, \mathsf{Male}, \mathsf{Father}\}.$$

Then $\mathcal{O}_2$ is an S-conservative extension of $\mathcal{O}_1$ w.r.t. $\mathcal{QL}_{\mathsf{CN}}$, but $\mathsf{Parent} \sqcap \mathsf{Male} \sqsubseteq \mathsf{Father}$ separates $\mathcal{O}_1$ and $\mathcal{O}_2$ w.r.t. $\mathcal{QL}_{\mathcal{ALC}}$. ◇

When choosing different DLs $\mathcal{L}$, the resulting notions of inseparability usually differ.

Example 9. Let

$$\mathcal{O}_1 = \{\mathsf{Human} \sqsubseteq \exists \mathsf{parent}.\top\}, \quad \mathcal{O}_2 = \{\mathsf{Human} \sqsubseteq \exists \mathsf{parent}.\mathsf{Male} \sqcap \exists \mathsf{parent}.\neg\mathsf{Male}\}$$

and $S = \{\mathsf{Human}, \mathsf{parent}\}$. Then $\mathcal{O}_1$ and $\mathcal{O}_2$ are S-inseparable w.r.t. $\mathcal{QL}_{\mathcal{ALC}}$, but $\mathsf{Human} \sqsubseteq \neg(\leq 1\, \mathsf{parent}\, \top)$ S-separates $\mathcal{O}_1$ and $\mathcal{O}_2$ w.r.t. $\mathcal{QL}_{\mathcal{ALCQ}}$. ◇

If $\mathcal{L}$ provides all Boolean operators, a useful *alternative definition* of $\mathcal{QL}_{\mathcal{L}}$ is the set of queries $C \sqsubseteq \bot$ with C an $\mathcal{L}$-concept, i.e., two ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$ are S-inseparable w.r.t. $\mathcal{QL}_{\mathcal{L}}$ iff there is no $\mathcal{L}_S$-concept C that is satisfiable w.r.t. $\mathcal{O}_1$ but not w.r.t. $\mathcal{O}_2$, or vice versa. To see that the definitions are equivalent, observe that $C \sqsubseteq D$ separates $\mathcal{O}_1$ and $\mathcal{O}_2$ iff $C \sqcap \neg D \sqsubseteq \bot$ separates $\mathcal{O}_1$ and $\mathcal{O}_2$.

The query languages $\mathcal{QL}_{\mathcal{L}}$ are often appropriate during the design process of an ontology.

Example 10. Assume an ontology designer extends a medical ontology $\mathcal{O}_0$ written in a DL $\mathcal{L}$ with a set $\mathcal{O}_1$ of $\mathcal{L}$-sentences in order to cover some part of medicine, say anatomy, in more detail. He wants to ensure that this operation does not impact the subsumptions between complex concepts built over symbols that are already defined in $\mathcal{O}_0$ and not related to anatomy. Then, he should ensure that $\mathcal{O}_0 \cup \mathcal{O}_1$ is an S-conservative extension of $\mathcal{O}_0$ w.r.t. $\mathcal{QL}_\mathcal{L}$, where S consists of all symbols from $\mathcal{O}_0$ that are unrelated to anatomy. Similarly, conservative extensions w.r.t. $\mathcal{QL}_\mathcal{L}$ can be used to ensure that *deletions* of sentences from an ontology do not change the subsumptions between concepts in an unexpected way. In the case that existing sentences are modified, the designer should use inseparability instead of conservative extensions. ◇

The query languages $\mathcal{QL}_\mathcal{L}$ are powerful enough to define useful notions of modularity, for example via Definitions 3 and 4. In particular, inseparability w.r.t. $\mathcal{QL}_\mathcal{L}$ is suited to ensure a relatively strong form of module independence, in contrast to inseparability w.r.t. $\mathcal{QL}_\perp$ and $\mathcal{QL}_{\mathsf{CN}}$. We now discuss the formal properties of $\mathcal{QL}_\mathcal{L}$ in more detail. We concentrate on giving an overview of available results and thus defer longer proofs to later sections or the appendix.

Robustness Properties

In many cases, $(\mathcal{L}, \mathcal{QL}_\mathcal{L})$ is robust under vocabulary extensions and joins. The following result is a consequence of Theorems 15 and 16 in Section 2.6.2.

Theorem 1. *Let $\mathcal{L}$ be any of the DLs $\mathcal{ALC}$, $\mathcal{ALCQ}$, $\mathcal{ALCI}$, $\mathcal{ALCQI}$, $\mathcal{ALCU}$, $\mathcal{ALCQU}$, $\mathcal{ALCIU}$, $\mathcal{ALCQIU}$. Then $(\mathcal{L}, \mathcal{QL}_\mathcal{L})$ is robust under vocabulary extensions and joins.*

Thus, standard constructors such as inverse roles and number restrictions do not cause any difficulties as far as robustness under vocabulary extensions and joins is concerned. The situation is different for nominals and role hierarchies.

Proposition 1. *$(\mathcal{ALCO}, \mathcal{QL}_{\mathcal{ALCO}})$ and $(\mathcal{ALCH}, \mathcal{QL}_{\mathcal{ALCH}})$ are not robust under vocabulary extensions and joins.*

Proof. We start with non-robustness under vocabulary extensions in $\mathcal{ALCO}$. Let

$$\mathcal{O}_1 = \{\top \sqsubseteq \exists r.\top\},$$
$$\mathcal{O}_2 = \mathcal{O}_1 \cup \{A \sqsubseteq \forall r.\neg A, \neg A \sqsubseteq \forall r.A\}.$$

Then $\mathcal{O}_2$ is a conservative extension of $\mathcal{O}_1$ w.r.t. $\mathcal{QL}_{\mathcal{ALCO}}$, thus $\mathcal{O}_1 \approx_S^{\mathcal{QL}_{\mathcal{ALCO}}} \mathcal{O}_2$ for $S = \{r\}$. Now observe that $\{a\} \sqsubseteq \forall r.\neg\{a\}$ separates the two ontologies w.r.t. $\mathcal{QL}_{\mathcal{ALCO}}$, for any nominal $\{a\}$. Thus, $\mathcal{O}_1 \not\approx_{S'}^{\mathcal{QL}_{\mathcal{ALCO}}} \mathcal{O}_2$ for $S' = S \cup \{a\}$. Observe that the nominal $\{a\}$ has no connection whatsoever with the two ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$.

Now consider $\mathcal{ALCH}$. Let

$$\mathcal{O}_1 = \{\top \sqsubseteq \forall r_i \forall r_j.\bot \mid i,j = 1,2\} \cup \{\exists r_1.\top \equiv \exists r_2.\top\},$$
$$\mathcal{O}_2 = \mathcal{O}_1 \cup \{s \sqsubseteq r_1, s \sqsubseteq r_2, \exists r_1.\top \sqsubseteq \exists s.\top\}.$$

Then $\mathcal{O}_2$ is a conservative extension of $\mathcal{O}_1$ w.r.t. $\mathcal{QL}_{\mathcal{ALCH}}$, thus $\mathcal{O}_1 \approx_S^{\mathcal{QL}_{\mathcal{ALCH}}} \mathcal{O}_2$ for $S = \{r_1, r_2\}$. However, $\exists r_1.\top \sqcap \forall r_1.A \sqsubseteq \exists r_2.A$ separates the two ontologies w.r.t. $\mathcal{QL}_{\mathcal{ALCH}}$, where A is a fresh concept name. The proof of non-robustness under joins is similar and deferred to Proposition 7 in Appendix A. □

In Section 2.4, we have argued that robustness under replacement is an essential property for most applications of inseparability, such as modularity. Theorem 1 thus indicates that one has to be rather careful when using $\mathcal{ALCO}$ and $\mathcal{ALCH}$ as a query language. In particular, weak and strong modules as in Definition 3 and 4 are inappropriate. However, it is possible to "build in" robustness under vocabulary extensions when defining a module:

Definition 6. *Let $\mathcal{O}_1 \subseteq \mathcal{O}_2$ be ontologies, S a signature, and $\mathcal{QL}$ a query language. Then $\mathcal{O}_1$ is a* weak S-module in $\mathcal{O}_2$ w.r.t. $\mathcal{QL}$ robust under vocabulary extensions *if $\mathcal{O}_2$ is an S'-conservative extension of $\mathcal{O}_1$ w.r.t. $\mathcal{QL}$ for all S' such that* $\mathsf{sig}(\mathcal{O}_1 \cup \mathcal{O}_2) \cap S' \subseteq S$. ◇

In a similar way, it is possible to define modules with robustness under joins built in.

We now turn to robustness under replacement. This property fails for many standard description logics such as the ones in the following result.

Theorem 2. *Let $\mathcal{L}$ be any of the DLs $\mathcal{ALC}, \mathcal{ALCQ}, \mathcal{ALCI}, \mathcal{ALCQI}$, $\mathcal{ALCO}$, $\mathcal{ALCHO}$. Then $(\mathcal{L}, \mathcal{QL}_{\mathcal{L}})$ is not robust under replacement.*

Proof. Let

$$\mathcal{O}_1 = \emptyset, \quad \mathcal{O}_2 = \{A \sqsubseteq \exists r.B\}, \quad S = \{A, B\}, \quad \mathcal{O} = \{A \equiv \top, B \equiv \bot\}.$$

It is not hard to see that for every first-order sentence φ with $\mathsf{sig}(\varphi) \subseteq S$,

$$\mathcal{O}_2 \models \varphi \text{ iff } \{\exists x\, A(x) \rightarrow \exists y\, B(y)\} \models \varphi.$$

However, the $\mathcal{FO}$-ontology $\{\exists x\, A(x) \rightarrow \exists y\, B(y)\}$ has no non-tautological consequences that can be formulated in $\mathcal{QL}_{\mathcal{L}}$ using only symbols from S. Hence, $\mathcal{O}_2$ is an S-conservative extension of $\mathcal{O}_1$ w.r.t. $\mathcal{QL}_{\mathcal{L}}$. But then, $\mathcal{O}_1 \cup \mathcal{O} \not\approx_S^{\mathcal{QL}_{\mathcal{L}}} \mathcal{O}_2 \cup \mathcal{O}$ since $\top \sqsubseteq \bot$ separates the two ontologies. □

In Section 2.4, we have argued that robustness under vocabulary extensions is needed for the reuse modules. Thus, Theorem 2 indicates that one has to be careful when using the query languages listed in the theorem for this application. For example, modules such as in Definitions 3, 4 and 6 are not appropriate. One approach to fix this is to define a notion of module that has robustness under replacements built in:

Definition 7. *Let $\mathcal{O}_1 \subseteq \mathcal{O}_2$ be ontologies, S a signature, and $\mathcal{QL}$ a query language. Then $\mathcal{O}_2$ is a* weak S-module of $\mathcal{O}_2$ w.r.t. $\mathcal{QL}$ robust under replacement *if for all ontologies $\mathcal{O}$ with* $\mathsf{sig}(\mathcal{O}) \cap \mathsf{sig}(\mathcal{O}_2) \subseteq S$*, $\mathcal{O}_2 \cup \mathcal{O}$ is an S-conservative extension of $\mathcal{O}_1 \cup \mathcal{O}$ w.r.t. $\mathcal{QL}$.* ◇

The next result identifies a second approach to deal with Theorem 2: if robustness under replacement is desired, switch from $(\mathcal{L}, \mathcal{QL}_{\mathcal{L}})$ to $(\mathcal{L}, \mathcal{QL}_{\mathcal{LU}})$. If this is done, robustness under replacements is recovered and it suffices to work with weak modules as in Definition 3. We shall see in Section 2.6 that the two approaches to deal with Theorem 2 are identical in some rather strong sense. For now, we only show that the addition of a universal role to the query language usually recovers robustness under replacement.

Theorem 3. *Let $\mathcal{L}$ be any of the DLs $\mathcal{ALC}$, $\mathcal{ALCI}$, $\mathcal{ALCQ}$, $\mathcal{ALCQI}$. Then $(\mathcal{LU}$, $\mathcal{QL}_{\mathcal{LU}})$, and thus also $(\mathcal{L}, \mathcal{QL}_{\mathcal{LU}})$, is robust under replacement.*

Proof. Let $\mathcal{O}_1 \approx_S^{\mathcal{QL}_\mathcal{L}} \mathcal{O}_2$ and assume that $\mathcal{O}_1 \cup \mathcal{O} \models C_0 \sqsubseteq D_0$, where $\mathsf{sig}(\mathcal{O}) \cap \mathsf{sig}(\mathcal{O}_1 \cup \mathcal{O}_2) \subseteq S$ and $\mathsf{sig}(C_0 \sqsubseteq D_0) \subseteq S$. Let $S' = S \cup \mathsf{sig}(\mathcal{O})$. By robustness under signature extensions, $\mathcal{O}_1 \approx_{S'}^{\mathcal{QL}_\mathcal{L}} \mathcal{O}_2$. Clearly

$$\mathcal{O}_1 \models \forall u. \bigsqcap_{C \sqsubseteq D \in \mathcal{O}} \neg C \sqcup D \sqsubseteq \forall u.(\neg C_0 \sqcup D_0).$$

Since $\mathcal{O}_1 \approx_{S'}^{\mathcal{QL}_\mathcal{L}} \mathcal{O}_2$, $\mathcal{O}_2$ entails the same subsumption. Thus $\mathcal{O}_2 \cup \mathcal{O} \models C_0 \sqsubseteq D_0$, as required. □

We present one additional observation regarding robustness under replacement. In the proof of Theorem 2, r is a role that does not occur in S. It turns out that such roles are required to obtain counterexamples to robustness under replacement. The following result is proved in Appendix A.

Theorem 4. *Let $\mathcal{L}$ be any of the DLs $\mathcal{ALC}$, $\mathcal{ALCI}$, $\mathcal{ALCQ}$, $\mathcal{ALCQI}$, S a signature, and assume that $\mathcal{O}_1$ and $\mathcal{O}_2$ contain only roles from S. Then*

$$\mathcal{O}_1 \approx_S^{\mathcal{QL}_\mathcal{L}} \mathcal{O}_2 \quad \textit{implies} \quad \mathcal{O}_1 \cup \mathcal{O} \approx_S^{\mathcal{QL}_\mathcal{L}} \mathcal{O}_2 \cup \mathcal{O},$$

for all $\mathcal{L}$-ontologies $\mathcal{O}$ with $\mathsf{sig}(\mathcal{O}) \cap \mathsf{sig}(\mathcal{O}_1 \cup \mathcal{O}_2) \subseteq S$.

Decision Problems

We now discuss the computational complexity of deciding inseparability and conservative extensions w.r.t. $\mathcal{QL}_\mathcal{L}$. The following result is due to [21, 36].

Theorem 5. *Let $\mathcal{L}$ be any of the DLs $\mathcal{ALC}$, $\mathcal{ALCI}$, $\mathcal{ALCQ}$ and $\mathcal{ALCQI}$. Then the S-inseparability problem for $(\mathcal{L}, \mathcal{QL}_\mathcal{L})$ is 2-*EXPTIME*-complete. Moreover, the S-conservativity and conservativity problem are 2-*EXPTIME*-complete as well.*

We remark that, in [21, 36], the results stated in Theorem 5 are proved for S-conservativity w.r.t. $\mathcal{QL}_\mathcal{L}$ for $\mathcal{ALC}$and $\mathcal{ALCQI}$ only. However, it is not too difficult to extend the proofs to $\mathcal{ALCI}$ and $\mathcal{ALCQI}$ and S-inseparability instead of S-conservativity. For the extensions of the logics $\mathcal{L}$ with the universal role we note the following conjecture.

Conjecture 1. Let $\mathcal{L}$ be any of the DLs $\mathcal{ALCU}$, $\mathcal{ALCIU}$, $\mathcal{ALCQU}$, $\mathcal{ALCQIU}$. Then the S-inseparability problem for $(\mathcal{L}, \mathcal{QL}_{\mathcal{L}})$ is 2-ExpTime-complete.

Interestingly, the addition of nominals to $\mathcal{ALCQI}$ leads to undecidability [36].

Theorem 6. *For $(\mathcal{ALCQIO}, \mathcal{QL}_{\mathcal{ALCQIO}})$, the S-inseparability problem and conservativity problem are undecidable.*

Recall that $\mathcal{LO}$ denotes the extension of a DL $\mathcal{L}$ by nominals. We conjecture that even with nominals, S-inseparability is still 2-ExpTime-complete for the DLs from above which include $\mathcal{ALC}$but are strictly below $\mathcal{ALCQI}$:

Conjecture 2. Let $\mathcal{L}$ be any of the DLs $\mathcal{ALC}$, $\mathcal{ALCI}$, $\mathcal{ALCQ}$ or their extension by the universal role. Then the S-inseparability problem for $(\mathcal{LO}, \mathcal{QL}_{\mathcal{LO}})$ is 2-ExpTime-complete.

Notions of a module such as the ones given in Definitions 6 and 7 give rise to other decision problems than the S-inseparability problem and the conservativity problem. The following result addresses the case of Definition 6. It is proved in [16] for $(\mathcal{ALCO}, \mathcal{QL}_{\mathcal{ALCO}})$. The proof is easily extended to the other listed cases.

Theorem 7. *Let $\mathcal{L}$ be any of the DLs $\mathcal{ALC}$, $\mathcal{ALCI}$, $\mathcal{ALCQ}$ or their extension by the universal role. Given two $\mathcal{LO}$-ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$ and a signature S, it is undecidable whether $\mathcal{O}_1$ is a weak modules of $\mathcal{O}_2$ w.r.t. $\mathcal{QL}_{\mathcal{LO}}$ robust under replacement.*

2.5.4 ABoxes and Conjunctive Queries

If ontologies are used together with an ABox, inseparability based on subsumption is usually too weak. In this section, we define notions of inseparability based on instance checking and conjunctive query answering.

In principle, there are two ways to include the ABox into our framework: as part of the ontology and as part of the query language. The first option is appropriate if the ontology and ABox are closely coupled, e.g. they are designed and maintained together, and the ABox does not change significantly more often than the ontology. In this case, we should define S-inseparability between *knowledge bases.*

Definition 8. Let $\mathcal{L}$ be a description logic. We call knowledge bases $\mathcal{K}_1$ and $\mathcal{K}_2$

a) *S-inseparable w.r.t. $\mathcal{L}$-instance checking* iff, for any $\mathcal{L}$-assertion $C(a)$ with $\mathsf{sig}(C) \subseteq S$, we have $\mathcal{K}_1 \models C(a)$ iff $\mathcal{K}_2 \models C(a)$.
b) *S-inseparable w.r.t. $\mathcal{L}$-conjunctive queries* iff, for all instantiated $\mathcal{L}$-conjunctive queries $q = \exists \mathbf{v}.\varphi(\mathbf{v}, \mathbf{a})$ with $\mathsf{sig}(q) \subseteq S$, we have $\mathcal{K}_1 \models q$ iff $\mathcal{K}_2 \models q$. ◇

It is easy to see that the latter implies the former. However, the converse is false. For a simple example, let $\mathcal{L} = \mathcal{ALC}$,

$$\mathcal{O}_1 = \emptyset, \quad \mathcal{O}_2 = \{A \sqsubseteq \exists r.B\}, \quad \mathcal{A}_1 = \mathcal{A}_2 = \{A(a)\}, \text{ and } S = \{A, B\}.$$

Then $\mathcal{K}_1$ and $\mathcal{K}_2$ are S-inseparable w.r.t. instance checking, but the conjunctive query $\exists x.B(x)$ separates the two knowledge bases. When applied to knowledge bases with an empty ABox, S-inseparability w.r.t. $\mathcal{L}$-instance checking and (subsumption-based) S-inseparability w.r.t. $\mathcal{QL}_{\mathcal{L}}$ coincide. To see this, note that (i) if $C \sqsubseteq D$ separates $\mathcal{K}_1$ and $\mathcal{K}_2$, then so does $D \sqcup \neg C(a)$, for any $a \in \mathsf{N_C}$; conversely (ii) if $C(a)$ separates $\mathcal{K}_1$ and $\mathcal{K}_2$, then so does $\top \sqsubseteq C$.

It should be clear that the above two notions of inseparability fit into our framework: knowledge bases can be translated into sets of $\mathcal{FO}$-sentences, and thus can be viewed as ontologies. However, in most applications that use ABoxes, the above approach does not seem appropriate because the ontology (conceptual modelling) and the ABox (actual data) have a different status. In particular, the ABox is usually unknown when the ontology is developed and changes much more frequently than the ontology. This observation suggests that it is useful to consider a notion of inseparability that captures ABoxes as unknown "black boxes" and thus quantifies over *all possible ABoxes*. This corresponds to making the ABox part of the query, instead of the ontology.

Definition 9. Let $\mathcal{L}$ be a description logic. We say that ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$ are

c) *S-inseparable w.r.t. $\mathcal{L}$-instance checking* iff, for all $\mathcal{L}$-ABoxes $\mathcal{A}$ with $\mathsf{sig}(\mathcal{A}) \subseteq S$ and $\mathcal{L}$-assertions $C(a)$ with $\mathsf{sig}(C) \subseteq S$, we have $(\mathcal{O}_1, \mathcal{A}) \models C(a)$ iff $(\mathcal{O}_2, \mathcal{A}) \models C(a)$.
d) *S-inseparable w.r.t. $\mathcal{L}$-conjunctive queries* iff, for all $\mathcal{L}$-ABoxes $\mathcal{A}$ with $\mathsf{sig}(\mathcal{A}) \subseteq S$ and all instantiated $\mathcal{L}$-conjunctive queries $q = \exists \mathbf{v}.\varphi(\mathbf{v}, \mathbf{a})$, we have $(\mathcal{O}_1, \mathcal{A}) \models q$ iff $(\mathcal{O}_2, \mathcal{A}) \models q$. $\diamond$

Observe that c)-d) apply to ontologies, in contrast to a)-b), which apply to knowledge bases. Again, it is easy to see that if two ontologies are S-inseparable w.r.t. $\mathcal{L}$-conjunctive queries, then they are S-inseparable w.r.t. $\mathcal{L}$-instance checking. The converse fails, with essentially the same argument as for a) and b) above.

Note that S-inseparability of two ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$ w.r.t. $\mathcal{L}$-instance checking implies S-inseparability w.r.t. $\mathcal{QL}_{\mathcal{L}}$ because, as above, if $C \sqsubseteq D$ separates $\mathcal{O}_1$ and $\mathcal{O}_2$, then so does the empty ABox together with the instance query $D \sqcup \neg C(a)$. However, unlike for a), the conserve does not hold.

Example 11. Let

$$\mathcal{O}_1 = \emptyset, \quad \mathcal{O}_2 = \mathcal{O}_1 \cup \{A \sqsubseteq \forall r.\neg A, \neg A \sqsubseteq \forall r.A\}, \quad S = \{r\}.$$

Then $\mathcal{O}_1$ and $\mathcal{O}_2$ are S-inseparable w.r.t. $\mathcal{QL}_{\mathcal{ALC}}$, but $(\mathcal{O}_1, \{r(a,a)\})$ is consistent and $(\mathcal{O}_2, \{r(a,a)\})$ is inconsistent. Thus, the ABox $\mathcal{A} = \{r(a,a)\}$ and assertion $\bot(a)$ separate $\mathcal{O}_1$ and $\mathcal{O}_2$. Note that this example is similar to the counterexample given in the proof of Proposition 1. $\diamond$

S-inseparability w.r.t. $\mathcal{L}$-instance checking is stronger than S-inseparability w.r.t. $\mathcal{QL}_{\mathcal{L}}$ because of the availability of the ABox, which allows us to fix a part of the model up to isomorphism.

We show that the notions of inseparability given under c) and d) live inside our framework. In what follows, we use individual names as first-order constants. For an $\mathcal{L}$-ABox $\mathcal{A}$, define a corresponding first-order sentence

$$\mathcal{A}^\sharp := \bigwedge_{C(a)\in\mathcal{A}} C^\sharp[x/a] \wedge \bigwedge_{r(a,b)\in\mathcal{A}} r(a,b).$$

For an instantiated conjunctive query $\exists \mathbf{v}.\varphi(\mathbf{v},\mathbf{a})$, define a corresponding first-order sentence $q^\sharp$ by replacing each concept atom $C(a)$ in φ with $C^\sharp[x/a]$.

Definition 10. Let $\mathcal{L}$ be a description logic. Then

- $\mathcal{QL}^I_\mathcal{L}$ is the set of first-order sentences $\mathcal{A}^\sharp \rightarrow C^\sharp[x/a]$, where $\mathcal{A}$ is an $\mathcal{L}$-ABox, C an $\mathcal{L}$-concept and q an individual name;
- $\mathcal{QL}^q_\mathcal{L}$ is the set of first-order sentences $\mathcal{A}^\sharp \rightarrow q^\sharp$, where $\mathcal{A}$ is an $\mathcal{L}$-ABox and q is an instantiated $\mathcal{L}$-conjunctive query. $\Diamond$

It is easy to see that two ontologies are S-inseparable w.r.t. $\mathcal{L}$-instance checking in the sense of Definition 9 c) iff they are S-inseparable w.r.t. $\mathcal{QL}^I_\mathcal{L}$ and that they are S-inseparable w.r.t. $\mathcal{L}$-conjunctive queries in the sense of Definition 9 d) iff they are S inseparable w.r.t. $\mathcal{QL}^q_\mathcal{L}$.

As indicated by the discussion above, query languages such as $\mathcal{QL}^I_\mathcal{L}$ and $\mathcal{QL}^q_\mathcal{L}$ induce an even stronger notion of S-inseparability than $\mathcal{QL}_\mathcal{L}$. Intuitively, their power is between that of $\mathcal{QL}_\mathcal{L}$ and $\mathcal{QL}_\mathcal{FO}$, as discussed in Section 2.5.5 below. For $\mathcal{QL}^I_\mathcal{L}$ and $\mathcal{QL}^q_\mathcal{L}$, neither the computational complexity of deciding inseparability nor robustness properties have been investigated for description logics extending $\mathcal{ALC}$. In Section 2.8, we will discuss a number of results for these query languages for ontologies based on weak DLs such as $\mathcal{EL}$ and DL-Lite.

2.5.5 Semantic Inseparability, First- and Second-Order Queries

The query languages $\mathcal{QL}$ considered so far are all proper fragments of first-order logic. In fact, it is not difficult to show that inseparability w.r.t. $\mathcal{QL}_\mathcal{FO}$, the set of all first-order sentences, is stronger than any of them. To see this, consider

$$\mathcal{O}_1 = \emptyset, \quad \mathcal{O}_2 = \{A \sqsubseteq \exists r.(A \sqcap B) \sqcap \exists r.(A \sqcap \neg B)\}, \quad S = \{A\}$$

Then, $\mathcal{O}_2$ is an S-conservative extension of $\mathcal{O}_1$ w.r.t. all query languages considered so far, but the first-order sentence $\exists x.A(x) \Rightarrow \exists y.(x \neq y \wedge A(y))$ separates $\mathcal{O}_1$ and $\mathcal{O}_2$.

Theorem 8. *Let $\mathcal{L}$ be a fragment of $\mathcal{FO}$. Then $(\mathcal{L}, \mathcal{QL}_\mathcal{FO})$ is robust under vocabulary extensions, joins and under replacement.*

Proof. Robustness under vocabulary extensions and joins follow from Theorems 15 and 16 in Section 2.6.2. Robustness under replacement follows from robustness under vocabulary extensions and the fact that first-order logic is closed under Boolean operators: from $\mathcal{O}_1 \approx^{\mathcal{FO}}_S \mathcal{O}_2$, it follows that for all $\mathcal{L}$-ontologies $\mathcal{O}$ with $\mathsf{sig}(\mathcal{O}) \cap \mathsf{sig}(\mathcal{O}_1 \cup \mathcal{O}_2) \subseteq S$ and $\mathcal{FO}$-sentences φ with $\mathsf{sig}(\varphi) \subseteq S$, we have

$$\mathcal{O}_1 \models \bigwedge \mathcal{O} \rightarrow \varphi \quad \Leftrightarrow \quad \mathcal{O}_2 \models \bigwedge \mathcal{O} \rightarrow \varphi,$$

which implies

$$\mathcal{O}_1 \cup \mathcal{O} \models \varphi \quad \Leftrightarrow \quad \mathcal{O}_2 \cup \mathcal{O} \models \varphi.$$

$\square$

An even stronger query language than $\mathcal{QL}_{\mathcal{FO}}$ is $\mathcal{QL}_{\mathcal{SO}}$, the set of all second-order sentences. As has already been mentioned, S-inseparability w.r.t. $\mathcal{QL}_{\mathcal{SO}}$ implies inseparability w.r.t. *any* query language allowed in our framework. Interestingly, inseparability w.r.t. $\mathcal{QL}_{\mathcal{SO}}$ is equivalent to a semantic notion of inseparability, which we introduce next. Given a model $\mathcal{I}$ and a signature S, we denote by $\mathcal{I}_{|S}$ the *S-reduct* of $\mathcal{I}$; i.e., $\Delta^{\mathcal{I}_{|S}} = \Delta^{\mathcal{I}}$, $X^{\mathcal{I}_{|S}} = X^{\mathcal{I}}$ for all $X \in S$, and predicates not in S are not interpreted by $\mathcal{I}_{|S}$.

Definition 11. Let $\mathcal{O}_1, \mathcal{O}_2$ be ontologies and S a signature. We say that

- $\mathcal{O}_1$ and $\mathcal{O}_2$ are *semantically S-inseparable* and write $\mathcal{O}_1 \approx_S^{\mathsf{sem}} \mathcal{O}_2$ if
$$\{\mathcal{I}_{|S} \mid \mathcal{I} \models \mathcal{O}_1\} = \{\mathcal{I}_{|S} \mid \mathcal{I} \models \mathcal{O}_2\}.$$
- $\mathcal{O}_2$ is a *semantic S-conservative extension* of $\mathcal{O}_1$ if $\mathcal{O}_1 \subseteq \mathcal{O}_2$ and $\mathcal{O}_1 \approx_S^{\mathsf{sem}} \mathcal{O}_2$. ◇

We simply speak of a *semantic conservative extension* if $S = \mathsf{sig}(\mathcal{O}_1)$. In the literature, the term 'model conservative extension' is used synonymously. We now show equivalence of semantic inseparability and inseparability w.r.t. $\mathcal{QL}_{\mathcal{SO}}$.

Theorem 9. *Let $\mathcal{O}_1$ and $\mathcal{O}_2$ be ontologies and S a signature. Then the following conditions are equivalent:*

- $\mathcal{O}_1 \approx_S^{\mathcal{QL}_{\mathcal{SO}}} \mathcal{O}_2$;
- $\mathcal{O}_1 \approx_S^{\mathsf{sem}} \mathcal{O}_2$.

Proof. The implication from Point 1 to Point 2 follows from the fact that no second-order sentence using only predicates from S can distinguish two models whose reducts to S are isomorphic. The other direction holds since we have $\mathcal{O}_1 \approx_S^{\mathsf{sem}} \mathcal{O}_2$ if $\mathcal{O}_1$ implies the second-order sentence $\exists S_1 \cdots \exists S_n . \bigwedge \mathcal{O}_2$ with $\{S_1, \ldots, S_n\} = \mathsf{sig}(\mathcal{O}_2) \setminus S$ and $\mathcal{O}_2$ implies the second-order sentence $\exists S_1' \cdots \exists S_m' . \bigwedge \mathcal{O}_1$ with $\{S_1', \ldots, S_m'\} = \mathsf{sig}(\mathcal{O}_1) \setminus S$. □

The following robustness properties are easily proved by exploiting the high expressive power of $\mathcal{SO}$.

Theorem 10. *Let $\mathcal{L}$ be a fragment of $\mathcal{SO}$. Then $(\mathcal{L}, \mathcal{QL}_{\mathcal{SO}})$ is robust under vocabulary extensions, joins and under replacement.*

The relation between inseparability w.r.t. $\mathcal{QL}_{\mathcal{FO}}$ and semantic inseparability has extensively been discussed in the literature on software specification [25, 39, 46]. In [8, 47], the reader can find a proof that semantic inseparability is strictly stronger than inseparability w.r.t. $\mathcal{QL}_{\mathcal{FO}}$. However, when restricted to *finite models*, the two notions are equivalent because every finite model can be described up to isomorphism in first-order logic with equality. The computational complexity of deciding semantic S-inseparability has been studied in [36, 27].

Theorem 11. *For $\mathcal{ALC}$-ontologies, semantic conservativity is Π_1^1-hard, thus neither decidable nor recursively enumerable.*

Conditions under which semantic inseparability becomes decidable are investigated in [27]. For example, if S contains no roles, then semantic S-inseparability becomes decidable:

Theorem 12. *For $\mathcal{ALC}$-ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$ and signatures S containing no roles, it is decidable (and* NExpTime$^{\text{NP}}$*-complete) whether $\mathcal{O}_1 \approx_S^{\mathsf{sem}} \mathcal{O}_2$.*

2.6 More on Robustness

We take a closer look at the robustness properties and relate them to standard notions from logic such as interpolation and Robinson consistency.

2.6.1 Robustness under Replacement

We start with addressing the question why $(\mathcal{FO}, \mathcal{QL}_{\mathcal{FO}})$ and $(\mathcal{LU}, \mathcal{QL}_{\mathcal{LU}})$, with $\mathcal{L}$ as in Theorem 3, are robust under replacement while $(\mathcal{L}, \mathcal{QL}_{\mathcal{L}})$ are not. A first hint is given by the simple proof of Theorem 8, which crucially exploits that the query language $\mathcal{QL}_{\mathcal{FO}}$ is closed under the Boolean operators. This is clearly not the case for $\mathcal{QL}_{\mathcal{L}}$, but also not for $\mathcal{QL}_{\mathcal{LU}}$. However, it is well-known that there is a close connection between the universal role and the Boolean operators, see e.g. [20]. To make this connection formal, we extend $\mathcal{QL}_{\mathcal{L}}$ to the query language $\mathcal{QL}_{\mathcal{L}}^B$ that consists of all *Boolean combinations* of $\mathcal{QL}_{\mathcal{L}}$-sentences: $\mathcal{QL}_{\mathcal{L}}^B$ is the set of sentences ψ defined by

$$\psi ::= \varphi \mid \psi_1 \wedge \psi_2 \mid \neg\psi,$$

where $\varphi \in \mathcal{QL}_{\mathcal{L}}$ and ψ, ψ_1, ψ_2 range over $\mathcal{QL}_{\mathcal{L}}^B$-sentences. The next result explains why $(\mathcal{LU}, \mathcal{QL}_{\mathcal{LU}})$ is robust under replacement. It is an easy consequence of the well-known fact that $\mathcal{QL}_{\mathcal{LU}}$ and $\mathcal{QL}_{\mathcal{L}}^B$ have the same expressive power in the following sense [20]: for every $\varphi \in \mathcal{QL}_{\mathcal{LU}}$, there exists a $\varphi^* \in \mathcal{QL}_{\mathcal{L}}^B$ with $\mathsf{sig}(\varphi) = \mathsf{sig}(\varphi^*)$ such that $\mathcal{I} \models \varphi$ iff $\mathcal{I} \models \varphi^*$ holds for all interpretations $\mathcal{I}$, and vice versa.

Theorem 13. *Let $\mathcal{L}$ be any of the DLs $\mathcal{ALC}$, $\mathcal{ALCI}$, $\mathcal{ALCQ}$, $\mathcal{ALCQI}$. Then $\approx_S^{\mathcal{QL}_{\mathcal{L}}^B} = \approx_S^{\mathcal{QL}_{\mathcal{LU}}}$ for any signature S.*

In Section 2.5.3, two approaches have been identified to deal with non-robustness under replacements of the query language $\mathcal{QL}_{\mathcal{L}}$ when $\mathcal{L}$ does not contain the universal role: either build robustness under replacements into the definition of a module as in Definition 7 or switch from $(\mathcal{L}, \mathcal{QL}_{\mathcal{L}})$ to $(\mathcal{L}, \mathcal{QL}_{\mathcal{LU}})$. We now show that these two approaches are actually identical. To do this, it is convenient to define a notion of inseparability that has robustness under replacements build in, analogous to Definition 7: $\mathcal{QL}_{\mathcal{L}}^O$ consists of all sentences

$$(\bigwedge \mathcal{O}^\sharp) \rightarrow \varphi,$$

where $\mathcal{O}$ is an $\mathcal{L}$-ontology and φ a query from $\mathcal{QL}_{\mathcal{L}}$.

Theorem 14. *Let $\mathcal{L}$ be any of the DLs $\mathcal{ALC}$, $\mathcal{ALCI}$, $\mathcal{ALCQ}$, $\mathcal{ALCQI}$. Then $\approx_S^{\mathcal{QL}^O_\mathcal{L}} = \approx_S^{\mathcal{QL}_{\mathcal{LU}}}$ for any signature S.*

Proof. By Theorem 13, it suffices to show that $\approx_S^{\mathcal{QL}^O_\mathcal{L}} = \approx_S^{\mathcal{QL}^B_\mathcal{L}}$. The 'if' direction is easy since it is not hard to see that $\mathcal{QL}^O_\mathcal{L} \subseteq \mathcal{QL}^B_\mathcal{L}$. For the 'only if' direction, assume $\mathcal{O}_1 \approx_S^{\mathcal{QL}^O_\mathcal{L}} \mathcal{O}_2$. Assume that $\mathcal{O}_2 \models \varphi$, where φ is a Boolean combination of $\mathcal{L}_S$-implications. Clearly, φ is equivalent to a conjunction of formulas of the form

$$(\top \sqsubseteq C_0) \vee \cdots \vee (\top \sqsubseteq C_m) \vee \neg(\top \sqsubseteq C_{m+1}) \vee \cdots \vee \neg(\top \sqsubseteq C_n)$$

and each such conjunct is equivalent to

$$(\top \sqsubseteq C_0) \vee \cdots \vee (\top \sqsubseteq C_m) \vee \neg(\top \sqsubseteq D),$$

where $D = C_{m+1} \sqcap \cdots \sqcap C_n$. Moreover, using the fact that models of $\mathcal{L}$-ontologies are closed under disjoint unions, one can show that there exists $i \leq m$ such that $\mathcal{O}_2 \models (\top \sqsubseteq C_i) \vee \neg(\top \sqsubseteq D)$, which implies $\mathcal{O}_2 \cup \{\top \sqsubseteq D\} \models \top \sqsubseteq C_i$. From $\mathcal{O}_1 \approx_S^{\mathcal{QL}^O_\mathcal{L}} \mathcal{O}_2$, we obtain $\mathcal{O}_1 \cup \{\top \sqsubseteq D\} \models \top \sqsubseteq C_i$. Since this holds for all conjuncts of φ, we get $\mathcal{O}_1 \models \varphi$. □

Now, the two mentioned approaches are identical since it clearly follows from Theorem 14 that $\mathcal{O}_1$ is a weak S-module w.r.t. $\mathcal{QL}_\mathcal{L}$ robust under replacement iff $\mathcal{O}_1$ is a weak S-module w.r.t. $\mathcal{QL}_{\mathcal{LU}}$ according to Definition 3.

Finally, Theorem 14 can provide us with another interesting perspective on the relationship between $(\mathcal{L}, \mathcal{QL}_\mathcal{L})$ and $(\mathcal{L}, \mathcal{QL}_{\mathcal{LU}})$. Obviously, $\approx_S^{\mathcal{QL}_{\mathcal{LU}}} \subseteq \approx_S^{\mathcal{QL}_\mathcal{L}}$. However, $\approx_S^{\mathcal{QL}_{\mathcal{LU}}}$ is much more than just *some* coarsening of $\approx_S^{\mathcal{QL}_\mathcal{L}}$ that yields robustness under replacement: it is the maximal one. The following is a direct consequence of Theorem 14 and the definition of $\mathcal{QL}^O_\mathcal{L}$.

Corollary 1. *Let $\mathcal{L}$ be any of the DLs $\mathcal{ALC}$, $\mathcal{ALCI}$, $\mathcal{ALCQ}$, $\mathcal{ALCQI}$ and S a signature. Then $\approx_S^{\mathcal{QL}_{\mathcal{LU}}}$ is the maximal subset of $\approx_S^{\mathcal{QL}_\mathcal{L}}$ such that for all $\mathcal{L}$-ontologies $\mathcal{O}_1$, $\mathcal{O}_2$ and $\mathcal{O}$ with $\mathsf{sig}(\mathcal{O}) \cap \mathsf{sig}(\mathcal{O}_1 \cup \mathcal{O}_2) \subseteq S$, $\mathcal{O}_1 \approx_S^{\mathcal{QL}_\mathcal{L}} \mathcal{O}_2$ implies $\mathcal{O}_1 \cup \mathcal{O} \approx_S^{\mathcal{QL}_\mathcal{L}} \mathcal{O}_2 \cup \mathcal{O}$.*

2.6.2 Robustness under Vocabulary Extensions and Interpolation

In this section, we discuss robustness under vocabulary extensions and its relationship to interpolation. We consider the following standard notion of interpolation.

Definition 12 (Interpolation). A query language $\mathcal{QL}$ has *weak interpolation* iff for every set Ψ of $\mathcal{QL}$-sentences and every $\mathcal{QL}$-sentence φ such that $\Psi \models \varphi$, there exists a set $I(\Psi, \varphi)$ of $\mathcal{QL}$-sentence such that

- $\mathsf{sig}(I(\Psi, \varphi)) \subseteq \mathsf{sig}(\Psi) \cap \mathsf{sig}(\varphi)$;
- $\Psi \models I(\Psi, \varphi)$;
- $I(\Psi, \varphi) \models \varphi$.

$\mathcal{QL}$ has interpolation if there always exists a *finite* set $I(\Psi, \varphi)$ with these properties. ◇

For any *compact*[1] query language $\mathcal{QL}$, weak interpolation implies interpolation. Interpolation has been investigated extensively in mathematical logic and for modal logics closely related to DLs. For example, propositional logic, first- and second-order logic, and basic modal logic have interpolation [13, 26]. Proofs of interpolation for a variety of DLs are given in Appendix B.

The following proposition shows that weak interpolation implies robustness under vocabulary extensions:

Proposition 2. *Suppose $\mathcal{L}$ and $\mathcal{QL}$ are given and $\mathcal{L} \subseteq \mathcal{QL}$. If $\mathcal{QL}$ has weak interpolation, then $(\mathcal{L}, \mathcal{QL})$ is robust under vocabulary extensions.*

Proof. Suppose $\mathcal{O}_1 \approx_S^{\mathcal{QL}} \mathcal{O}_2$ and let φ be a $\mathcal{QL}$-sentence with $\mathsf{sig}(\varphi) \cap \mathsf{sig}(\mathcal{O}_1 \cup \mathcal{O}_2) \subseteq S$ such that $\mathcal{O}_1 \models \varphi$. By weak interpolation, there exists an interpolant $I(\mathcal{O}_1, \varphi)$. From $\mathsf{sig}(I(\mathcal{O}_1, \varphi)) \subseteq S$ we obtain $\mathcal{O}_2 \models I(\mathcal{O}_1, \varphi)$. Hence $\mathcal{O}_2 \models \varphi$. □

Theorem 15. *Let $\mathcal{L}$ be any of the DLs $\mathcal{ALC}$, $\mathcal{ALCQ}$, $\mathcal{ALCI}$, $\mathcal{ALCQI}$, $\mathcal{ALCU}$, $\mathcal{ALCQU}$, $\mathcal{ALCIU}$, $\mathcal{ALCQIU}$. The following are robust under vocabulary extensions:*

1. *$(\mathcal{L}, \mathcal{QL}_{\mathcal{L}})$;*
2. *$(\mathcal{L}', \mathcal{QL}_{\mathcal{FO}})$, for any fragment $\mathcal{L}'$ of first-order logic;*
3. *$(\mathcal{L}', \mathcal{QL}_{\mathcal{SO}})$, for any fragment $\mathcal{L}'$ of second-order logic.*

Proof. By Proposition 9 of Appendix B, the mentioned languages $\mathcal{QL}_{\mathcal{L}}$ have interpolation and it thus remains to apply Proposition 2 to establish Point 1. Points 2 and 3 follow from Proposition 2 and the fact that first-and second order logic have interpolation.

We state a partial converse of Proposition 2. An *infinitary ontology* is a finite or infinite set of second-order sentences.

Proposition 3. *Suppose $(\mathcal{QL}, \mathcal{QL})$ is robust under vocabulary extensions for infinitary ontologies. Then $\mathcal{QL}$ has weak interpolation.*

Proof. Assume $\mathcal{O} \models \varphi$, where $\mathcal{O}$ is a set of $\mathcal{QL}$-sentences and φ a $\mathcal{QL}$-sentence. Set $S = \mathsf{sig}(\mathcal{O}) \cap \mathsf{sig}(\varphi)$ and

$$\mathcal{O}' = \{\psi \in \mathcal{QL} \mid \mathcal{O} \models \psi, \mathsf{sig}(\psi) \subseteq S\}.$$

Then $\mathcal{O} \models \mathcal{O}'$ and $\mathcal{O}$ and $\mathcal{O}'$ are S-inseparable w.r.t. $\mathcal{QL}$. By robustness under vocabulary extensions for infinitary ontologies, $\mathcal{O}$ and $\mathcal{O}'$ are S'-inseparable w.r.t. $\mathcal{QL}$, where $S' = \mathsf{sig}(\varphi)$. From $\mathcal{O} \models \varphi$ we obtain $\mathcal{O}' \models \varphi$. Hence $I(\mathcal{O}, \varphi) = \mathcal{O}'$ is as required. □

[1] $\mathcal{QL}$ is compact if $\Psi \models \varphi$ implies that there exists a finite subset Ψ' of Ψ such that $\Psi' \models \varphi$. First-order logic and its fragments are compact. Second-order logic and first-order logic over finite models are not compact.

An interesting logic which is robust under vocabulary extension but does not have interpolation is first-order logic *over finite models*. One can easily show weak interpolation for this logic, and, therefore, robustness under vocabulary extensions. The argument for failure of interpolation is as follows: using a binary predicate symbol $<$ and a unary predicate symbol red, one can write a finite set $\mathcal{O}_1$ of first-order sentences that is satisfied exactly in those finite models which have an even number of points (state that $<$ is a linear order, exactly every second point is red, and the first and last point have distinct colours). Use a different binary relation symbol $<'$ and unary predicate green for a finite set $\mathcal{O}_2$ of first-order axioms which is satisfied exactly in finite models with an odd number of points. Then $\mathcal{O}_1 \models_{\mathsf{fin}} \neg \bigwedge \mathcal{O}_2$, but there does not exist a finite interpolant.

2.6.3 Robustness under Joins and Interpolation

We discuss the relation between robustness under joins and interpolation. For $(\mathcal{QL}_{\mathcal{FO}}, \mathcal{QL}_{\mathcal{FO}})$, robustness under joins is easily seen to be equivalent to the well-known Robinson joint consistency property [13]: if T_1 and T_2 are consistent first-order theories both extending an S-complete theory T_0 with $\mathsf{sig}(T_0) \subseteq S$ (i.e. $T_0 \models \varphi$ or $T_0 \models \neg\varphi$ for all φ over S) and $\mathsf{sig}(T_1) \cap \mathsf{sig}(T_2) \subseteq S$, then $T_1 \cup T_2$ is consistent. As this property is known to be closely related to interpolation, it is no surprise that robustness under joins is closely related to interpolation as well.

Proposition 4. *Let $\mathcal{L} \subseteq \mathcal{QL}$ and assume that $\mathcal{QL}$ is closed under Boolean operators. If $\mathcal{QL}$ has weak interpolation, then $(\mathcal{L}, \mathcal{QL})$ is robust under joins.*

Proof. Suppose $\mathcal{QL}$ has weak interpolation, $\mathcal{O}_1$ and $\mathcal{O}_2$ are S-inseparable w.r.t. $\mathcal{QL}$ and $\mathsf{sig}(\mathcal{O}_1) \cap \mathsf{sig}(\mathcal{O}_2) \subseteq S$. Assume $\mathcal{O}_1 \cup \mathcal{O}_2 \models \varphi$ where $\mathsf{sig}(\varphi) \subseteq S$. Then

$$\mathcal{O}_1 \models \bigwedge \mathcal{O}_2 \rightarrow \varphi.$$

Take an interpolant $I(\mathcal{O}_1, \bigwedge \mathcal{O}_2 \rightarrow \varphi)$ and observe that its signature is contained in S. Then $\mathcal{O}_2 \models I(\mathcal{O}_1, \bigwedge \mathcal{O}_2 \rightarrow \varphi)$, by S-inseparability of $\mathcal{O}_1$ and $\mathcal{O}_2$ w.r.t. $\mathcal{QL}$ and the assumption that $\mathcal{QL}$ is closed under Boolean operators. Hence $\mathcal{O}_2 \models \bigwedge \mathcal{O}_2 \rightarrow \varphi$. But then $\mathcal{O}_2 \models \varphi$. □

Theorem 16. *Let $\mathcal{L}$ be any of the DLs $\mathcal{ALC}$, $\mathcal{ALCQ}$, $\mathcal{ALCI}$, $\mathcal{ALCQI}$, $\mathcal{ALCU}$, $\mathcal{ALCIU}$, $\mathcal{ALCQU}$, $\mathcal{ALCQIU}$. The following are robust under joins.*

1. *$(\mathcal{L}, \mathcal{QL}_{\mathcal{L}})$;*
2. *$(\mathcal{L}', \mathcal{QL}_{\mathcal{FO}})$, for any fragment $\mathcal{L}'$ of first-order logic;*
3. *$(\mathcal{L}', \mathcal{QL}_{\mathcal{SO}})$, for any fragment $\mathcal{L}'$ of second-order logic.*

Proof. Using Proposition 4, Points 2 and 3 follow from the fact that first- and second-order logic have interpolation and are closed under Boolean operators. For $\mathcal{ALC}$, $\mathcal{ALCQ}$, $\mathcal{ALCI}$ and $\mathcal{ALCQI}$, robustness under joins is proved in Proposition 10 of Appendix A.

It remains to show robustness under joins for $\mathcal{LU}$ if $\mathcal{L}$ is any of $\mathcal{ALC}$, $\mathcal{ALCI}$, $\mathcal{ALCQ}$, $\mathcal{ALCQI}$. By Proposition 9 of Appendix B, $\mathcal{QL}_{\mathcal{LU}}$ has interpolation. As $\mathcal{QL}_{\mathcal{LU}}$ and $\mathcal{QL}^B_{\mathcal{L}}$ have the same expressive power (see Section 2.6.1), $\mathcal{QL}^B_{\mathcal{L}}$ has interpolation. Hence, by Proposition 4, $(\mathcal{QL}^B_{\mathcal{L}}, \mathcal{QL}^B_{\mathcal{L}})$ is robust under joins. Once more since $\mathcal{QL}_{\mathcal{LU}}$ and $\mathcal{QL}^B_{\mathcal{L}}$ have the same expressive power, $(\mathcal{LU}, \mathcal{QL}_{\mathcal{LU}})$ is robust under joins. □

The following is a partial converse of Proposition 4.

Proposition 5. *Let $\mathcal{QL}$ be a fragment of first-order logic closed under Boolean operators such that $(\mathcal{QL}, \mathcal{QL})$ is robust under joins for infinitary ontologies. Then $\mathcal{QL}$ has interpolation.*

Proof. Suppose $\mathcal{O} \models \varphi$. Let $S = \mathsf{sig}(\mathcal{O}) \cap \mathsf{sig}(\varphi)$ and

$$\Psi_0 = \{\psi \in \mathcal{QL} \mid \mathsf{sig}(\psi) \subseteq S, \mathcal{O} \models \psi\}.$$

We show that Ψ_0 is an interpolant for $(\mathcal{O}, \varphi)$. Assume not. Then $\Psi_0 \cup \{\neg\varphi\}$ is satisfiable. Take a model $\mathcal{I}$ satisfying $\Psi_0 \cup \{\neg\varphi\}$ and denote by $\mathcal{O}'$ the set of $\mathcal{QL}$-sentences ψ with $\mathsf{sig}(\psi) \subseteq S$ which are true in $\mathcal{I}$. Then both $\mathcal{O}' \cup \mathcal{O}$ and $\mathcal{O}' \cup \{\neg\varphi\}$ are S-conservative extensions of $\mathcal{O}'$ w.r.t. $\mathcal{QL}$. By robustness under joins, $\mathcal{O}' \cup \mathcal{O} \cup \{\neg\varphi\}$ is an S-conservative extension of $\mathcal{O}'$ w.r.t. $\mathcal{QL}$; in particular, it is consistent. Hence $\mathcal{O} \not\models \varphi$ and we have derived a contradiction. □

2.7 Uniform Interpolation and Forgetting

Assume that we want to re-use the information that an ontology $\mathcal{O}$ provides about a certain signature S in an application where only queries formulated in $\mathcal{QL}$ are relevant. Then we have two options. The first is to extract an S-module, i.e., to identify a subset $\mathcal{O}'$ of $\mathcal{O}$ that is S-inseparable from $\mathcal{O}$ w.r.t. $\mathcal{QL}$ (or satisfies even stronger conditions, cf. the other possible definitions of a module). In this approach, which is pursued in [27, 14], the extracted module may also contain symbols not in S. The second option is to construct a new ontology $\mathcal{O}_S$ that *contains only symbols from S* and has the same consequences in $\mathcal{QL}$ over S as $\mathcal{O}$ in the sense that $\mathcal{O}$ and $\mathcal{O}_S$ are S-inseparable w.r.t. $\mathcal{QL}$. Whether such an ontology $\mathcal{O}_S$ exists and can be effectively constructed depends on the ontology language used, the signature S and the query language $\mathcal{QL}$. This problem has been studied by different research communities under various names such as *forgetting* [33, 42, 49], *uniform interpolation* [41, 22, 48] and *variable elimination*.

Definition 13 (Uniform Interpolation (Forgetting)). $(\mathcal{L}, \mathcal{QL})$ *has* uniform interpolation *if for every $\mathcal{L}$-ontology $\mathcal{O}$ and signature S there exists an $\mathcal{L}$-ontology $\mathcal{O}_S$ with* $\mathsf{sig}(\mathcal{O}_S) \subseteq S$ *such that $\mathcal{O} \approx^{\mathcal{QL}}_{S'} \mathcal{O}_S$ for all S' with* $\mathsf{sig}(\mathcal{O}) \cap S' \subseteq S$. *In this case, $\mathcal{O}_S$ is called a S-*uniform interpolant *of $\mathcal{O}$ w.r.t. $(\mathcal{L}, \mathcal{QL})$.* ◇

Note that if $(\mathcal{QL}, \mathcal{QL})$ has uniform interpolation, then $\mathcal{QL}$ has interpolation: assume that $\mathcal{O} \models \varphi$ and let $S = \mathsf{sig}(\mathcal{O}) \cap \mathsf{sig}(\varphi)$. Then $\mathcal{O} \models \mathcal{O}_S$ and $\mathcal{O}_S \models \varphi$. Thus, $\mathcal{O}_S$ is an interpolant. This explains the name *uniform* interpolant as it is an interpolant that does not depend on the right hand side of the consequence.

In the context of forgetting, it can also be sensible to define uniform interpolation in a slightly less strict way, namely by demanding inseparability only w.r.t. S rather than the signatures S' used in Definition 13. By Proposition 2, this definition is equivalent to the stronger one if $\mathcal{QL}$ has interpolation.

There is an intimate connection between the computation of S-uniform interpolants and deciding S-inseparability. In particular, if $(\mathcal{L}, \mathcal{QL})$ has uniform interpolation, then S-inseparability of $\mathcal{L}$-ontologies $\mathcal{O}$ and $\mathcal{O}'$ w.r.t. $\mathcal{QL}$ can be decided by first computing S-uniform interpolants $\mathcal{O}_S$ of $\mathcal{O}$ w.r.t. $(\mathcal{L}, \mathcal{QL})$ and $\mathcal{O}'_S$ of $\mathcal{O}'$ w.r.t. $(\mathcal{L}, \mathcal{QL})$, and then checking whether $\mathcal{O}'_S \models \varphi$ for all $\varphi \in \mathcal{O}_S$ and $\mathcal{O}_S \models \varphi$ for all $\varphi \in \mathcal{O}'_S$.

It is easy to see that $(\mathcal{QL}_{\mathcal{SO}}, \mathcal{QL}_{\mathcal{SO}})$ has uniform interpolation: given an ontology $\mathcal{O}$ in second-order logic and a signature S, let $\mathcal{O}_S$ consist of the sentence

$$\exists S_1. \cdots \exists S_n. \bigwedge \mathcal{O},$$

where $\{S_1, \dots, S_n\} = \mathsf{sig}(\mathcal{O}) \setminus S$. Then $\mathcal{O}_S$ is a uniform interpolant. We now investigate uniform interpolants for weaker languages.

Example 12. Let

$$\mathcal{O} = \{\mathsf{Hand} \sqsubseteq \mathsf{Body\_part}, \mathsf{Body\_part} \sqsubseteq \mathsf{Physical\_object}\}$$

and $S = \{\mathsf{Hand}, \mathsf{Physical\_object}\}$. Then $\mathcal{O}_S = \{\mathsf{Hand} \sqsubseteq \mathsf{Physical\_object}\}$ is an S-uniform interpolant of $\mathcal{O}$ w.r.t. any of the query languages $\mathcal{QL}$ considered in this paper. ◇

Example 13. Let

$$\mathcal{O} = \{\mathsf{Human} \sqsubseteq \exists \mathsf{child\_of}.\mathsf{Male}\}, \quad S = \{\mathsf{Human}, \mathsf{Male}\}$$

Then $\mathcal{O}_S = \{\mathsf{Human} \sqsubseteq \exists u.\mathsf{Male}\}$ is an S-uniform interpolant of $\mathcal{O}$ w.r.t. any query language with expressivity between $\mathcal{QL}_{\mathcal{ALCU}}$ and $\mathcal{QL}_{\mathcal{SO}}$. ◇

Theorem 17. *Let $\mathcal{L}$ be any of the DLs $\mathcal{ALC}$, $\mathcal{ALCQ}$, $\mathcal{ALCI}$, $\mathcal{ALCQI}$, $\mathcal{ALCU}$, $\mathcal{ALCIU}$, $\mathcal{ALCQU}$, $\mathcal{ALCQIU}$. Then for every $\mathcal{L}$-ontology $\mathcal{O}$ and every signature S that consists of concept names, there exists an S-uniform interpolant of $\mathcal{O}$ w.r.t. $(\mathcal{L}, \mathcal{QL}_{\mathcal{L}})$.*

Proof. By Theorem 15, all $(\mathcal{L}, \mathcal{QL}_{\mathcal{L}})$ are robust under vocabulary extensions. Thus, it is sufficient to show that there is an ontology $\mathcal{O}_S$ with $\mathcal{O} \approx_S^{\mathcal{QL}_{\mathcal{L}}} \mathcal{O}_S$. The set

$$\mathcal{O}'_S = \{C \sqsubseteq D \mid \mathcal{O} \models C \sqsubseteq D,\ C, D\ \mathcal{L}_S\text{-concepts}\}.$$

is our starting point. It has the required property, but may be infinite.

Assume first that $\mathcal{L}$ does not contain the universal role. Then, since S does not contain any roles, every concept C from $\mathcal{O}'_S$ is equivalent to some Boolean expression over S. As there are only finitely many non-equivalent such expressions, we obtain a finite set $\mathcal{O}_S \subseteq \mathcal{O}'_S$ which is S-inseparable from $\mathcal{O}'_S$ w.r.t. $\mathcal{QL}_\mathcal{L}$. $\mathcal{O}_S$ is as required.

Now assume that $\mathcal{L}$ contains the universal role. Then every $\mathcal{L}_S$-concept S can be regarded as a formula of modal logic S5. It is known (and straightforward to prove) that there are only finite many non-equivalent S5-formulas in a given number of variables. Hence, again, we obtain a finite set $\mathcal{O}_S \subseteq \mathcal{O}'_S$ which is S-inseparable from $\mathcal{O}'_S$ w.r.t. $\mathcal{QL}_\mathcal{L}$. $\mathcal{O}_S$ is as required. □

It seems worthwhile to point out that uniform interpolants may be large: the smallest such $\mathcal{O}_S$ from Theorem 17 can be of size exponential in $\mathcal{O}$ [28]. In [28], it is proved that none of the combinations $(\mathcal{L}, \mathcal{QL}_\mathcal{L})$ from Theorem 17 has uniform interpolation.

Theorem 18. *Let $\mathcal{L}$ be any of the DLs $\mathcal{ALC}$, $\mathcal{ALCQ}$, $\mathcal{ALCI}$, $\mathcal{ALCQI}$, $\mathcal{ALCU}$, $\mathcal{ALCIU}$, $\mathcal{ALCQU}$, $\mathcal{ALCQIU}$. Then $(\mathcal{L}, \mathcal{QL}_\mathcal{L})$ does not have uniform interpolation.*

Proof. For the $\mathcal{ALC}$-ontology $\mathcal{O}$ and signature S given in the proof of Lemma 6 of [28] there does not exist a $\mathcal{FO}$-ontology $\mathcal{O}_S$ with $\mathsf{sig}(\mathcal{O}_S) \subseteq S$ such that $\mathcal{O}$ and $\mathcal{O}_S$ are S-inseparable w.r.t. $\mathcal{QL}$, for any of the query languages $\mathcal{QL}$ listed above. □

It follows from the proof of Theorem 18 that $(\mathcal{QL}_\mathcal{FO}, \mathcal{QL}_\mathcal{FO})$ does not have uniform interpolation. We still provide a direct proof.

Proposition 6. *$(\mathcal{QL}_\mathcal{FO}, \mathcal{QL}_\mathcal{FO})$ does not have uniform interpolation.*

Proof. Let $\mathcal{O}$ be an axiomatisation of the theory of dense linear-orders using the binary relation symbol $<$. Each model of $\mathcal{O}$ has an infinite domain, $\mathcal{O}$ therefore implies $\varphi_n = \exists x_1 \cdots \exists x_n \bigwedge_{i \neq j} x_i \neq x_j$, for all $n \geq 1$. But there does not exist a finite and consistent set of first-order axioms over the empty signature which implies all φ_n, $n \geq 1$. □

We have seen that standard DLs and first-order logic do not have uniform interpolation. This defect can be cured by adding second-order quantification. We briefly discuss the extension $\mathcal{ALC}\mu$ of $\mathcal{ALC}$that has uniform interpolation. $\mathcal{ALC}\mu$-concepts C are defined as follows. Let V be an infinite set of concept variables. Then

- every $\mathcal{ALC}$-concept, possibly with some concept names replaced with concept variables, is an $\mathcal{ALC}\mu$-concept;
- if C is an $\mathcal{ALC}\mu$-concept in which $X \in V$ occurs positively (under an even number of negations), then $\mu X.C$ is an $\mathcal{ALC}\mu$-concept.

To assign a semantics to $\mathcal{ALC}\mu$-concepts, an interpretation is combined with an assignment $\tau : V \to 2^{\Delta^\mathcal{I}}$. Then the extension $(\mu X.C)^{\mathcal{I},\tau}$ of $\mu X.C$ is defined as

$$(\mu X.C)^{\mathcal{I},\tau} = \bigcap \{S \subseteq \Delta^\mathcal{I} \mid C^{\mathcal{I},\tau'} \subseteq S, \tau'(X) = S, \text{for all } Y \neq X\text{: } \tau'(Y) = \tau(Y)\}.$$

A *closed* $\mathcal{ALC}\mu$-concept is a $\mathcal{ALC}\mu$-concept without free concept variables. An $\mathcal{ALC}\mu$-ontology is a finite set of implications $C \sqsubseteq D$, where C and D are closed

$\mathcal{ALC}\mu$-concepts. Other notions are now defined in the same way as for $\mathcal{ALC}$. $\mathcal{ALC}\mu$ is a very powerful description logic in which subsumption is still ExpTime-complete. We refer the reader to [7, 3] for further information.

The following result can now be proved using uniform interpolation results for the modal μ-calculus from [18].

Theorem 19. *$(\mathcal{ALC}\mu, \mathcal{QL}_{\mathcal{ALC}\mu})$ has uniform interpolation.*

2.8 Weaker Description Logics and Acyclic TBoxes

So far, we have concentrated on extensions of $\mathcal{ALC}$and ontologies that are sets of implications between concepts or even first- and second-order sentences. In this section, we have a brief look at what happens if we consider weaker DLs and/or a weaker form of ontology called an acyclic ontology.

2.8.1 $\mathcal{EL}$

$\mathcal{EL}$ and its extensions form a family of lightweight description logics that are popular for the formulation of large medical and biological ontologies such as SNOMED CT. Technically, $\mathcal{EL}$ is the fragment of $\mathcal{ALC}$that admits only the constructors $C \sqcap D$ and $\exists r.C$ and $\mathcal{EL}$-ontologies are finite sets of implications $C \sqsubseteq D$ between $\mathcal{EL}$-concepts C, D. In $\mathcal{EL}$, subsumption and a number of other relevant reasoning tasks can be solved in polynomial time [2]. Note that every $\mathcal{EL}$-ontology is satisfiable and thus subsumption is not reducible to satisfiability and the query language $\mathcal{QL}_\bot$ does not separate any $\mathcal{EL}$-ontologies.

In the following, we briefly summarise what is known about modularity of $\mathcal{EL}$-ontologies. The query languages $\mathcal{QL}_{\mathcal{EL}}$, $\mathcal{QL}^I_{\mathcal{EL}}$, and $\mathcal{QL}^q_{\mathcal{EL}}$ are defined in the same way as the corresponding query languages for $\mathcal{ALC}$, except that all involved concepts have to be formulated in $\mathcal{EL}$. The following theorem summarises the results for inseparability in $\mathcal{EL}$ obtained in [37, 28, 38]:

Theorem 20

(i) S-inseparability w.r.t. $\mathcal{QL}^I_{\mathcal{EL}}$ coincides with S-inseparability w.r.t. $\mathcal{QL}_{\mathcal{EL}}$ but does not coincide with S-inseparability w.r.t. $\mathcal{QL}^q_{\mathcal{EL}}$.
(ii) For $\mathcal{QL}$ any of $\mathcal{QL}_{\mathcal{EL}}$, $\mathcal{QL}^I_{\mathcal{EL}}$, $\mathcal{QL}^q_{\mathcal{EL}}$:

- *S-inseparability w.r.t. $\mathcal{QL}$ is* ExpTime*-complete.*
- *$(\mathcal{EL}, \mathcal{QL})$ is robust under signature extensions and joins, but not under replacement.*
- *$(\mathcal{EL}, \mathcal{QL})$ does not have uniform interpolation.*

(iii) Semantic conservativity is undecidable for $\mathcal{EL}$-ontologies.

Many interesting problems remain open for $\mathcal{EL}$. For example, nothing is known about the minimal query language extending $\mathcal{QL}_{\mathcal{EL}}$ and robust under replacement.

2.8.2 DL-Lite

The DL-Lite family of description logics consists of lightweight languages whose main application is to describe constraints over data repositories. In contrast to other DLs, data complexity of query answering is within LOGSPACE for most members of the family, and conjunctive queries over ontologies and ABoxes can be effectively rewritten as SQL queries so that standard database query engines can be used for query answering [9, 10, 1].

Modularity properties and the complexity of corresponding reasoning problems have been investigated in [29, 30] for the dialects *DL-Lite$_{bool}$* and *DL-Lite$_{horn}$*. It turns out that those languages are rather well-behaved. We summarise here the behaviour of *DL-Lite$_{bool}$* and refer to [30] for information about *DL-Lite$_{horn}$*. *DL-Lite$_{bool}$* concepts are constructed from $\mathsf{N_C}$ and $\mathsf{N_R}$ using the Boolean operators, $\sqcap$ and $\neg$, and *unqualified* number restrictions $(\geq\ n\ r)$ and $(\leq\ n\ r)$, where r is a role name or its inverse. A *DL-Lite$_{bool}$* ontology is a finite set of implications between DL-Lite$_{bool}$-concepts. The query languages $\mathcal{QL}_{\text{DL-Lite}_{bool}}$ and $\mathcal{QL}^q_{\text{DL-Lite}_{bool}}$ are defined in the same way as the corresponding query languages for $\mathcal{ALC}$, except that now all concepts involved range over *DL-Lite$_{bool}$* concepts. The following theorem summarises what is known about *DL-Lite$_{bool}$* [30]:

Theorem 21. *Let* $(\mathcal{L}, \mathcal{QL}) = (\textit{DL-Lite}_{bool}, \mathcal{QL}_{\text{DL-Lite}_{bool}})$. *Then the following holds:*

- S*-inseparability w.r.t.* $\mathcal{QL}$ *is* Π^p_2*-complete;*
- $(\mathcal{L}, \mathcal{QL})$ *is robust under vocabulary extensions and joins, but not under replacement;*
- $(\mathcal{L}, \mathcal{QL})$ *has uniform interpolation.*

Let $(\mathcal{L}, \mathcal{QL}) = (\textit{DL-Lite}_{bool}, \mathcal{QL}^q_{\text{DL-Lite}_{bool}})$. *Then the following holds:*

- S*-inseparability w.r.t.* $\mathcal{QL}$ *is* Π^p_2*-complete;*
- $(\mathcal{L}, \mathcal{QL})$ *is robust under vocabulary extensions, joins and replacement.*

For results on uniform interpolation of $(\textit{DL-Lite}, \mathcal{QL}^q_{\text{DL-Lite}_{bool}})$, we once more refer to [30]. Experimental results on deciding S-inseparability using QBF-solvers are also reported in [30].

2.8.3 Acyclic Ontologies

For a description logic $\mathcal{L}$, an *acyclic* $\mathcal{L}$*-ontology* $\mathcal{O}$ is a finite set of expressions $A \equiv C$ and $A \sqsubseteq C$, A a concept name, such that

- no concept name occurs twice on the left hand side and
- the relation $\prec_{\mathcal{O}} \subseteq \mathsf{N_C} \times \mathsf{N_C}$, defined by $(A, B) \in \prec_{\mathcal{O}}$ iff B occurs in C for some $A \equiv C \in \mathcal{O}$ or $A \sqsubseteq C \in \mathcal{O}$, is acyclic.

We refer the reader to [3] for more information. Many ontologies from practical applications are acyclic, including such prominent cases as SNOMED CT. In this section, we discuss the impact on S-inseparability and modularity of switching from general ontologies as used so far to acyclic ones.

For members of the $\mathcal{EL}$ family of DLs, this switch can significantly reduce the complexity of reasoning about inseparability and may enable desirable features such as uniform interpolation. In particular, it has been shown that

- S-inseparability of acyclic $\mathcal{EL}$-ontologies w.r.t. $\mathcal{QL}_{\mathcal{EL}}$ is tractable, in contrast to ExpTime-hardness for general $\mathcal{EL}$-ontologies [28]. Experiments show that S-inseparability of distinct versions of the huge Snomed ct ontology can be swiftly decided in practice.
- deciding whether $\mathcal{O}_2$ is a semantic S-conservative extension of $\mathcal{O}_1$ is tractable for acyclic $\mathcal{EL}$-ontologies $\mathcal{O}_1, \mathcal{O}_2$ and signatures $S \supseteq \mathsf{sig}(\mathcal{O}_1)$, in contrast to undecidability for general $\mathcal{EL}$-ontologies [27]. An efficient module extraction algorithm based on this result has been implemented and successfully used with Snomed ct.

As already mentioned, another benefit of acyclic ontologies over general ones is uniform interpolation. While $(\mathcal{EL}, \mathcal{QL}_{\mathcal{EL}})$ does not admit uniform interpolation, an S-uniform interpolant w.r.t. $(\mathcal{EL}, \mathcal{QL}_{\mathcal{EL}})$ exists for every acyclic $\mathcal{EL}$-ontology $\mathcal{O}$ and signature S [28].

For expressive DL such as $\mathcal{ALC}$, acyclicity of ontologies typically does not yield any benefits. For example, it is still Π^1_1-hard to decide whether an acyclic $\mathcal{ALC}$-ontology is a semantic S-conservative extension of an empty ontology [27] and even for acyclic $\mathcal{ALC}$-ontologies and signatures S there does not always exist a uniform S-interpolant w.r.t. $(\mathcal{ALC}, \mathcal{QL}_{\mathcal{ALC}})$ [28].

2.9 Conclusion

We have investigated the notion of inseparability of ontologies w.r.t. a query language. As argued in the introduction and throughout the paper, this notion is central to logic-based approaches to modularity of ontologies. In particular, inseparability is commonly used to define independence of a module inside an ontology, and it can also be employed to understand and control the ramifications of re-using an ontology within another ontology. We have argued that the notion of inseparability has to be parameterised by a query language, and have identified three important meta-properties of inseparability: robustness under vocabulary extension, under joins, and under replacement. We have also investigated the relationship between these properties and interpolation, and discussed the computation complexity of deciding inseparability. Finally, be have briefly touched upon the relationship between inseparability and forgetting/uniform interpolation. Numerous technical problems are still open. To mention a few, the robustness properties and computational complexity of inseparability w.r.t. the query languages $\mathcal{QL}^q_{\mathcal{L}}$ defined in terms of conjunctive queries have not yet been investigated in any detail for DLs above $\mathcal{ALC}$. Also, 'positive' results for forgetting/uniform interpolation have been established only for very lightweight fragments of $\mathcal{ALC}$.

The theory developed in this paper has been evaluated in practice for ontologies formulated in the lightweight description logics $\mathcal{EL}$ and DL-Lite, with rather promising results [28, 30, 27]. In contrast, no 'practical' algorithms or experimental results have yet been obtained for deciding inseparability between ontologies formulated in more expressive languages such as $\mathcal{ALC}$. Thus, it remains to be explored whether deciding inseparability in such languages is feasible in practice, or whether more pragmatic approaches such as the locality-based one of [16] are the only feasible logic-based way to approach inseparability of ontologies formulated in expressive DLs.

We have confined our investigation to specific ontology and query languages, all of them fragments of second-order logic. In general, it would be interesting to develop a more general framework that allows to integrate ontologies formulated in (almost) arbitrary languages, covering, for example, non-classical logics, algebraic formalisms and non-monotonic languages. In software specification, the notion of institutions provides such a framework [24] and, recently, institutions have been proposed as a tool to investigate the modularity of ontologies [32, 35, 43]. However, a lot of work remains to be done. For example, the important distinction between query and ontology language has not yet been made explicit in the institutions approach.

References

1. Artale, A., Calvanese, D., Kontchakov, R., Zakharyaschev, M.: DL-Lite in the light of first-order logic. In: Proceedings of AAAI 2007, pp. 361–366. AAAI Press, Menlo Park (2007)
2. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope. In: Proceedings of IJCAI 2005, pp. 364–369. Professional Book Center (2005)
3. Baader, F., Calvanes, D., McGuiness, D., Nardi, D., Patel-Schneider, P.: The Description Logic Handbook: Theory, implementation and applications. Cambridge University Press, Cambridge (2003)
4. Bao, J., Caragea, D., Honavar, V.: On the semantics of linking and importing in modular ontologies. In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L.M. (eds.) ISWC 2006. LNCS, vol. 4273, pp. 72–86. Springer, Heidelberg (2006)
5. Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P., Stein, L.: OWL Web Ontology Language reference. W3C Recommendation (February 10, 2004)
6. Bouquet, P., Giunchiglia, F., van Harmelen, F., Serafini, L., Stuckenschmidt, H.: Contextualizing ontologies. Journal of Web Semantics 1(4), 325–343 (2004)
7. Bradfield, J., Stirling, C.: Modal mu-calculus. In: Blackburn, P., van Benthem, J., Wolter, F. (eds.) Handbook of Modal Logic. Elsevier, Amsterdam (2006)
8. Byers, P., Pitt, D.H.: Conservative extensions: a cautionary note. Bulletin of the EATCS 41, 196–201 (1990)
9. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: DL-Lite: Tractable description logics for ontologies. In: Proceedings of AAAI 2005, pp. 602–607. AAAI Press / MIT Press (2005)
10. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Data complexity of query answering in description logics. In: Proceedings of KR 2006, pp. 260–270. AAAI Press, Menlo Park (2006)

11. Calvanese, D., Eiter, T., Ortiz, M.: Answering regular path queries in expressive description logics: An automata-theoretic approach. In: Proceedings of AAAI 2007, pp. 391–396. AAAI Press, Menlo Park (2007)
12. ten Cate, B., Conradie, W., Marx, M., Venema, Y.: Definitorially complete description logics. In: Proceedings of KR 2006, pp. 79–89. AAAI Press, Menlo Park (2006)
13. Chang, C.C., Keisler, H.J.: Model Theory. Studies in Logic and the Foundations of Mathematics, vol. 73. Elsevier, Amsterdam (1990)
14. Cuenca Grau, B., Horrocks, I., Kazakov, Y., Sattler, U.: Just the right amount: Extracting modules from ontologies. In: Proceedings of WWW 2007, pp. 717–727. ACM, New York (2007)
15. Cuenca Grau, B., Horrocks, I., Kazakov, Y., Sattler, U.: A logical framework for modularity of ontologies. In: Proceedings of IJCAI 2007, pp. 298–303. AAAI Press, Menlo Park (2007)
16. Cuenca Grau, B., Horrocks, I., Kazakov, Y., Sattler, U.: Modular reuse of ontologies: Theory and practice. Journal of Artificial Intelligence Research 31, 273–318 (2008)
17. Cuenca Grau, B., Parsia, B., Sirin, E., Kalyanpur, A.: Modularity and web ontologies. In: Proceedings of KR 2006, pp. 198–209. AAAI Press, Menlo Park (2006)
18. D'Agostino, G., Hollenberg, M.: Uniform interpolation, automata, and the modal μ-calculus. In: Proceedings of AiML 1998, vol. 1. CSLI Publishers (1998)
19. Eiter, T., Fink, M., Woltran, S.: Semantical characterizations and complexity of equivalences in answer set programming. ACM Transactions in Computational Logic 8(3) (2007)
20. Gabbay, D.M., Kurucz, A., Wolter, F., Zakharyaschev, M.: Many-Dimensional Modal Logics: Theory and Applications. Studies in Logic and the Foundations of Mathematics, vol. 148. Elsevier, Amsterdam (2003)
21. Ghilardi, S., Lutz, C., Wolter, F.: Did I damage my ontology? A case for conservative extensions in description logic. In: Proceedings of KR 2006, pp. 187–197. AAAI Press, Menlo Park (2006)
22. Ghilardi, S., Zawadowski, M.: Undefinability of propositional quantifiers in the modal system S4. Studia Logica 55(2), 259–271 (1995)
23. Glimm, B., Lutz, C., Horrocks, I., Sattler, U.: Answering conjunctive queries in the $\mathcal{SHIQ}$ description logic. Journal of Artificial Intelligence Research 31, 150–197 (2008)
24. Goguen, J.A., Burstall, R.M.: Institutions: Abstract model theory for specification and programming. Journal of ACM 39(1), 95–146 (1992)
25. Goguen, J.A., Diaconescu, R., Stefaneas, P.: Logical support for modularisation. In: Proceedings of Logical Environments, pp. 83–130. Cambridge University Press, Cambridge (1993)
26. Goranko, V., Otto, M.: Model theory of modal logic. In: Blackburn, P., van Benthem, J., Wolter, F. (eds.) Handbook of Modal Logic. Elsevier, Amsterdam (2007)
27. Konev, B., Lutz, C., Walther, D., Wolter, F.: Semantic modularity and module extraction in description logics. In: Proceedings of ECAI 2008, pp. 55–59. IOS Press, Amsterdam (2008)
28. Konev, B., Walther, D., Wolter, F.: The logical difference problem for description logic terminologies. In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS, vol. 5195, pp. 259–274. Springer, Heidelberg (2008)
29. Kontchakov, R., Wolter, F., Zakharyaschev, M.: Modularity in DL-Lite. In: Proceedings of DL 2007, pp. 76–87. CEUR-WS (2007)
30. Kontchakov, R., Wolter, F., Zakharyaschev, M.: Can you tell the difference between DL-Lite ontologies. In: Proceedings of KR 2008, pp. 285–295. AAAI Press, Menlo Park (2008)

31. Kutz, O., Lutz, C., Wolter, F., Zakharyaschev, M.: E-connections of abstract description systems. Artificial Intelligence 156(1), 1–73 (2004)
32. Kutz, O., Mossakowski, T.: Conservativity in structured ontologies. In: Proceedings of ECAI 2008. IOS Press, Amsterdam (2008)
33. Lang, J., Liberatore, P., Marquis, P.: Propositional independence: formula-variable independence and forgetting. Journal of Artificial Intelligence Research 18, 391–443 (2003)
34. Lifschitz, V., Pearce, D., Valverde, A.: Strongly equivalent logic programs. ACM Transactions on Computational Logic 2(4), 526–541 (2001)
35. Lüttich, K., Masolo, C., Borgo, S.: Development of modular ontologies in CASL. In: Proceedings of WoMO 2006, CEUR-WS (2006)
36. Lutz, C., Walther, D., Wolter, F.: Conservative extensions in expressive description logics. In: Proceedings of IJCAI 2007, pp. 453–458. AAAI Press, Menlo Park (2007)
37. Lutz, C., Wolter, F.: Conservative extensions in the lightweight description logic $\mathcal{EL}$. In: Pfenning, F. (ed.) CADE 2007. LNCS, vol. 4603, pp. 84–99. Springer, Heidelberg (2007)
38. Lutz, C., Wolter, F.: Deciding inseparability and conservative extensions in the description logic $\mathcal{EL}$. Technical report, University of Liverpool (2008)
39. Maibaum, T.S.E.: Conservative extensions, interpretations between theories and all that! In: Bidoit, M., Dauchet, M. (eds.) TAPSOFT 1997. LNCS, vol. 1214, pp. 40–66. Springer, Heidelberg (1997)
40. Maksimova, L.: Complexity of interpolation and related problems in positive calculi. Journal of Symbolic Logic 67(1), 397–408 (2002)
41. Pitts, A.: On an interpretation of second-order quantification in first-order intuitionistic propositional logic. Journal of Symbolic Logic 57(1), 33–52 (1992)
42. Reiter, R., Lin, F.: Forget it! In: Proceedings of AAAI Fall Symposium on Relevance, pp. 154–159. AAAI Press, Menlo Park (1994)
43. Schorlemmer, M., Kalfoglou, Y.: Institutionalising ontology-based semantic integration. Applied Ontology (to appear, 2008)
44. Serafini, L., Borgida, A., Tamilin, A.: Aspects of distributed and modular ontology reasoning. In: Proceedings of IJCAI 2005, pp. 570–575. Professional Book Center (2005)
45. Spackman, K.: Managing clinical terminology hierarchies using algorithmic calculation of subsumption: Experience with SNOMED-RT. In: JAMIA (2000); Fall Symposium Special Issue
46. Veloso, P.: Yet another cautionary note on conservative extensions: a simple case with a computing flavour. EATCS-Bulletin 46, 188–193 (1992)
47. Veloso, P., Veloso, S.: Some remarks on conservative extensions. a socratic dialog. EATCS-Bulletin 43, 189–198 (1991)
48. Visser, A.: Uniform interpolation and layered bisimulation. In: Gödel 1996, Brno. Lecture Notes in Logic, vol. 6. Springer, Heidelberg (1996)
49. Wang, Z., Wang, K., Topor, R.W., Pan, J.Z.: Forgetting Concepts in DL-Lite. In: Proceedings of ESWC 2008, pp. 245–257. IOS Press, Amsterdam (2008)

A Deferred Proofs for Section 2.5

Proposition 7. *Neither $(\mathcal{ALCO}, \mathcal{QL}_{\mathcal{ALCO}})$ nor $(\mathcal{ALCH}, \mathcal{QL}_{\mathcal{ALCH}})$ is robust under joins.*

Proof. We start with $\mathcal{ALCO}$. Let $\mathcal{O}_0 = \{\top \sqsubseteq \exists r.\top\}$ and

$$\mathcal{O}_1 = \mathcal{O}_0 \cup \{A \sqsubseteq \forall r.\neg A, \neg A \sqsubseteq \forall r.A\}, \quad \mathcal{O}_2 = \mathcal{O}_0 \cup \{\{a\} \sqsubseteq \exists r.\{a\}\}.$$

Then both, $\mathcal{O}_1$ and $\mathcal{O}_2$, are conservative extensions of $\mathcal{O}_0$ w.r.t. $\mathcal{QL}_{\mathcal{ALCO}}$. On the other hand, $\mathcal{O}_1 \cup \mathcal{O}_2$ is inconsistent. It follows that $\mathcal{O}_1 \approx_S^{\mathcal{ALCO}} \mathcal{O}_2$ for $S = \{r\}$ but $\mathcal{O}_1 \cup \mathcal{O}_2 \not\approx_S^{\mathcal{ALCO}} \mathcal{O}_2$.

For $\mathcal{ALCH}$ take

$$\mathcal{O}_0 = \{\top \sqsubseteq \forall r_i \forall r_j.\bot \mid i, j = 1, 2\} \cup \{\exists r_1.\top \equiv \exists r_2.\top\},$$

$$\mathcal{O}_1 = \mathcal{O}_0 \cup \{s \sqsubseteq r_1, s \sqsubseteq r_2, \exists r_1.\top \sqsubseteq \exists s.\top\},$$

and

$$\mathcal{O}_2 = \mathcal{O}_0 \cup \{\exists r_1.\top \sqsubseteq \forall r_1.B \sqcap \forall r_2.\neg B\}$$

Then $\mathcal{O}_1$ and $\mathcal{O}_2$ are conservative extensions of $\mathcal{O}_0$ w.r.t. $\mathcal{QL}_{\mathcal{ALCH}}$. Hence $\mathcal{O}_1$ and $\mathcal{O}_2$ are S-inseparable w.r.t. $\mathcal{QL}_{\mathcal{ALCH}}$ for $S = \{r_1, r_2\}$.

On the other hand, $\mathcal{O}_1 \not\models \exists r_1 \bot \sqsubseteq \bot$ and $\mathcal{O}_1 \cup \mathcal{O}_2 \models \exists r_1.\top \sqsubseteq \bot$ and, therefore, $\mathcal{O}_1$ and $\mathcal{O}_1 \cup \mathcal{O}_2$ are nor S-inseparable w.r.t. $\mathcal{QL}_{\mathcal{ALCH}}$. □

Theorem 4. Let $\mathcal{L}$ be any of the DLs $\mathcal{ALC}$, $\mathcal{ALCI}$, $\mathcal{ALCQ}$, $\mathcal{ALCQI}$, S a signature, and assume that $\mathcal{O}_1$ and $\mathcal{O}_2$ contain only roles from S. Then

$$\mathcal{O}_1 \approx_S^{\mathcal{QL}_\mathcal{L}} \mathcal{O}_2 \quad \text{implies} \quad \mathcal{O}_1 \cup \mathcal{O} \approx_S^{\mathcal{QL}_\mathcal{L}} \mathcal{O}_2 \cup \mathcal{O},$$

for all $\mathcal{L}$-ontologies $\mathcal{O}$ with $\mathsf{sig}(\mathcal{O}) \cap \mathsf{sig}(\mathcal{O}_1 \cup \mathcal{O}_2) \subseteq S$.

Proof. We give a sketch for the case $\mathcal{L} = \mathcal{ALC}$. Let $\mathcal{O}_1 \approx_S^{\mathcal{QL}_\mathcal{L}} \mathcal{O}_2$ and assume that $\mathcal{O}_1 \cup \mathcal{O} \models C_0 \sqsubseteq D_0$, where $\mathsf{sig}(\mathcal{O}) \cap \mathsf{sig}(\mathcal{O}_1 \cup \mathcal{O}_2) \subseteq S$ and $\mathsf{sig}(C_0 \sqsubseteq D_0) \subseteq S$. Let $S' = S \cup \mathsf{sig}(\mathcal{O})$. By robustness under vocabulary extensions, $\mathcal{O}_1 \approx_{S'}^{\mathcal{QL}_\mathcal{L}} \mathcal{O}_2$. Let

$$\Gamma = \{\forall r_1. \cdots \forall r_n.(\neg C \sqcup D) \mid C \sqsubseteq D \in \mathcal{O}, r_i \in S', n \geq 0\}.$$

Using the condition that there are no additional roles in $\mathcal{O}_1$, it is not difficult to show that $d \in (\neg C_0 \sqcup D_0)^\mathcal{I}$ for every model $\mathcal{I}$ of $\mathcal{O}_1$ and $d \in \Delta^\mathcal{I}$ such that $d \in E^\mathcal{I}$ for all $E \in \Gamma$. By compactness, there exists a finite subset Γ' of Γ with the same property. It follows that

$$\mathcal{O}_1 \models \bigsqcap \Gamma' \sqcap C_0 \sqsubseteq D_0$$

from which we obtain $\mathcal{O}_2 \models \bigsqcap \Gamma' \sqcap C_0 \sqsubseteq D_0$. Since $\mathcal{O} \models \top \sqsubseteq \bigsqcap \Gamma'$, this implies $\mathcal{O}_2 \cup \mathcal{O} \models C_0 \sqsubseteq D_0$. □

B Interpolation

We provide a proof of the interpolation property and robustness under joins for basic DLs. Interpolation has been extensively investigated for many modal logics closely related to DLs [40], and also for some description logics [12]. However, one has to be careful when transferring interpolation results from modal logic to DL: most interpolation results in modal logic regard modal operators as logical symbols and thus not as a part of the signature. This implies that even if the input formulas Ψ or φ do not contain a modal operator $\Box$, this operator is nevertheless permitted in the interpolant for Ψ and φ. In DLs, the corresponding constructor $\forall r$ is not permitted in the interpolant unless the role r occurs in Ψ or φ.

Let $\mathcal{L}$ be a DL. For an interpretation $\mathcal{I}$, point $d \in \Delta^{\mathcal{I}}$ and signature S, we set

$$t_{\mathcal{I}}^{\mathcal{L},S}(d) = \{C \mid d \in C^{\mathcal{I}}, C \text{ an } \mathcal{L}_S\text{-concept}\}.$$

We say that two points d_1 and d_2 from possibly distinct interpretations $\mathcal{I}_1$ and $\mathcal{I}_2$ are $\mathcal{L}, S$-equivalent, written $(\mathcal{I}_1, d_1) \sim_S^{\mathcal{L}} (\mathcal{I}_2, d_2)$, if

$$t_{\mathcal{I}_1}^{\mathcal{L},S}(d_1) = t_{\mathcal{I}_2}^{\mathcal{L},S}(d_2).$$

We drop the $\mathcal{L}$ and write $d_1 \sim_S d_2$ instead of $(\mathcal{I}_1, d_1) \sim_S^{\mathcal{L}} (\mathcal{I}_2, d_2)$ if $\mathcal{I}_1, \mathcal{I}_2$ and $\mathcal{L}$ are understood. A mapping f from $\Delta^{\mathcal{I}_1}$ to $\Delta^{\mathcal{I}_2}$ is called *S-invariant* iff $x \sim_S f(x)$ for all x in the domain of f.

Proposition 8. *Let $\mathcal{L}$ be any of the DLs $\mathcal{ALC}$, $\mathcal{ALCQ}$, $\mathcal{ALCI}$, $\mathcal{ALCQI}$, $\mathcal{ALCU}$, $\mathcal{ALCQU}$, $\mathcal{ALCIU}$, or $\mathcal{ALCQIU}$. Let Ψ_1 and Ψ_2 be sets of $\mathcal{QL}_{\mathcal{L}}$-sentences and S a signature such that $\mathsf{sig}(\Psi_1) \cap \mathsf{sig}(\Psi_2) \subseteq S$. Assume there are models $\mathcal{I}_1$ and $\mathcal{I}_2$ of Ψ_1 and Ψ_2, respectively, such that*

(a) for all $\mathcal{L}_S$-concepts C: $C^{\mathcal{I}_1} \neq \emptyset$ iff $C^{\mathcal{I}_2} \neq \emptyset$.

Then there exists a model $\mathcal{I}$ of $\Psi_1 \cup \Psi_2$ such that

- *for all $\mathcal{L}_S$-concepts C: $C^{\mathcal{I}} \neq \emptyset$ iff $C^{\mathcal{I}_1} \neq \emptyset$ iff $C^{\mathcal{I}_2} \neq \emptyset$.*

If, in addition,

(b) there are points $d_1 \in \Delta^{\mathcal{I}_1}$ and $e_1 \in \Delta^{\mathcal{I}_2}$ such that $d_1 \sim_S^{\mathcal{L}} e_1$,

then $\mathcal{I}$ contains a point d satisfying

- *$t_{\mathcal{I}}^{\mathcal{L},S_1}(d) = t_{\mathcal{I}_2}^{\mathcal{L},S_1}(e_1)$, where $S_1 = S \cup ((\mathsf{N_C} \cup \mathsf{N_R}) \setminus \mathsf{sig}(\Psi_1))$ and*
- *$t_{\mathcal{I}}^{\mathcal{L},S_2}(d) = t_{\mathcal{I}_1}^{\mathcal{L},S_2}(d_1)$, where $S_2 = S \cup ((\mathsf{N_C} \cup \mathsf{N_R}) \setminus \mathsf{sig}(\Psi_2))$.*

Proof. Let $\mathcal{L}$ be any of the eight DLs listed in the proposition. We assume that $\mathcal{L}$ contains inverse roles (the prove for DLs without inverse rules is simpler and is easily obtained from the proof below). Let $\mathcal{I}_1$ and $\mathcal{I}_2$ be interpretations satisfying condition (a). We use a standard construction from model theory. Consider the disjoint union $\mathcal{I}_0$ of $\mathcal{I}_1$ and $\mathcal{I}_2$:

$$\mathcal{I}_0 = (\Delta^{\mathcal{I}_1} \cup \Delta^{\mathcal{I}_2}, A_1^{\mathcal{I}_0}, A_2^{\mathcal{I}_0}, \cdot^{\mathcal{I}_1}, \cdot^{\mathcal{I}_2}),$$

where we assume that $\Delta^{\mathcal{I}_1} \cap \Delta^{\mathcal{I}_2} = \emptyset$ and that $A_i^{\mathcal{I}_0} = \Delta^{\mathcal{I}_i}$ for fresh concept names A_i, $i = 1, 2$. Then $\mathcal{I}_0$ is elementarily equivalent to an interpretation $\mathcal{I}_0'$, which is countably recursively saturated [13]. This means that, if every finite subset of a recursive set of first-order formulas is satisfied in $\mathcal{I}_0'$, then the recursive set itself is satisfied in $\mathcal{I}_0'$. We will not go into the details of this construction but will use the following consequences: by taking, instead of $\mathcal{I}_1$ and $\mathcal{I}_2$, the corresponding substructures $\mathcal{I}_1'$ and $\mathcal{I}_2'$ (induced by $A_1^{\mathcal{I}_0'}$ and $A_2^{\mathcal{I}_0'}$) of $\mathcal{I}_0'$, we obtain interpretations which still satisfy condition (a) of Proposition 8 and have, in addition, the following properties:

1. For each $d_1 \in \Delta^{\mathcal{I}_1'}$, there exists $d_2 \in \Delta^{\mathcal{I}_2'}$ such that $(\mathcal{I}_1', d_1) \sim_S^{\mathcal{L}} (\mathcal{I}_2', d_2)$ and vice versa;
2. if $(\mathcal{I}_1', d_1) \sim_S^{\mathcal{L}} (\mathcal{I}_2', d_2)$ and $(d_1, e_1) \in r^{\mathcal{I}_1'}$ with $r \in S$, then there exists $e_2 \in \Delta^{\mathcal{I}_2'}$ with $(d_2, e_2) \in r^{\mathcal{I}_2'}$ and $(\mathcal{I}_1', e_1) \sim_S^{\mathcal{L}} (\mathcal{I}_2', e_2)$;
3. if $(\mathcal{I}_1', d_1) \sim_S^{\mathcal{L}} (\mathcal{I}_2', d_2)$ and $(d_2, e_2) \in r^{\mathcal{I}_2'}$ with $r \in S$, then there exists $e_1 \in \Delta^{\mathcal{I}_1'}$ with $(d_1, e_1) \in r^{\mathcal{I}_1'}$ and $(\mathcal{I}_1', e_1) \sim_S^{\mathcal{L}} (\mathcal{I}_2', e_2)$;
4. if $(\mathcal{I}_1', d_1) \sim_S^{\mathcal{L}} (\mathcal{I}_2', d_2)$ and $r \in S \cup S^-$ and $\mathcal{L}$ contains *number restrictions*, then there exists an S-invariant bijection between $\{d \mid (d_1, d) \in r^{\mathcal{I}_1'}\}$ and $\{e \mid (d_2, e) \in r^{\mathcal{I}_2'}\}$.

Intuitively, for Point 4 above, we need number restrictions in $\mathcal{L}$, because, without them, DLs are too weak to determine the number of r-successors of a node. In what follows, we use $\mathcal{I}_1$ and $\mathcal{I}_2$ to denote $\mathcal{I}_1'$ and $\mathcal{I}_2'$, respectively. If condition (b) of Proposition 8 is satisfied, then take d_1 and e_1 satisfying (b). Otherwise, take arbitrary S-equivalent points $d_1 \in \Delta^{\mathcal{I}_1}$ and $e_1 \in \Delta^{\mathcal{I}_2}$, which exist by Point 1 above. We unravel the model $\mathcal{I}_1$ starting from d_1 as follows: take infinitely many copies d^i, $i \geq 0$, of each $d \in \Delta^{\mathcal{I}_1}$ and define $\mathcal{J}_1$ by taking as the domain $\Delta^{\mathcal{J}_1}$ the set of all finite sequences

$$(d_1, r_2, d_2^{i_2}, r_3, d_3^{i_3}, \cdots, r_n, d_n^{i_n}),$$

where $d_i \in \Delta^{\mathcal{I}_1}$, $r_i \in \mathsf{N_R} \cup \mathsf{N_R}^- \cup \{\delta\}$ (with δ being some fresh "dummy" relation symbol) and $i_j \geq 0$ for $j \geq 2$, such that the following conditions hold:

(i) $i_j = 0$ whenever $r_j \in \mathsf{N_R} \cup \mathsf{N_R}^-$ and $\mathcal{L}$ contains qualified number restrictions;
(ii) $(d_i, d_{i+1}) \in r_{i+1}^{\mathcal{I}_1}$ whenever $r_i \in \mathsf{N_R} \cup \mathsf{N_R}^-$;
(iii) $d_i \neq d_{i+2}$ whenever $r_{i+1} = (r_{i+2})^-$ and $\mathcal{L}$ contains qualified number restrictions.

The interpretation function $\cdot^{\mathcal{J}_1}$ of $\mathcal{J}_1$ is defined as follows:

- $(d_1, r_2, d_2^{i_2}, \cdots, r_n, d_n^{i_n}) \in A^{\mathcal{J}_1}$ iff $d_n \in A^{\mathcal{I}_1}$, for all $A \in \mathsf{N_C}$;
- for all $r \in \mathsf{N_R}$, $r^{\mathcal{J}_1}$ consists of all pairs

  $$((d_1, r_2, d_2^{i_2}, \cdots, r_n, d_n^{i_n}), (d_1, r_2, d_2^{i_2}, \cdots, r_n, d_n^{i_n}, r_{n+1}, d_n^{i_{n+1}})) \in \Delta^{\mathcal{J}_1} \times \Delta^{\mathcal{J}_1},$$

  where $r_{n+1} = r$, and

  $$((d_1, r_2, d_2^{i_2}, \cdots, r_n, d_n^{i_n}, r_{n+1}, d_n^{i_{n+1}}), (d_1, r_2, d_2^{i_2}, \cdots, r_n, d_n^{i_n})) \in \Delta^{\mathcal{J}_1} \times \Delta^{\mathcal{J}_1}$$

  with $r_{n+1} = r^-$.

It is not difficult to show, that $(d_1, r_2, d_2^{i_2}, \cdots, r_n, d_n^{i_n}) \sim_{\mathsf{N_C} \cup \mathsf{N_R}} d_n$, for every $d_n \in \Delta^{\mathcal{I}_1}$. Observe that conditions (i) and (iii) ensure that, if $\mathcal{L}$ contains qualified number restrictions, then the number of r-successors of any point in $\Delta^{\mathcal{I}_1}$, $r \in \mathsf{N_C} \cup \mathsf{N_C}^-$, satisfying a certain set of concepts remains the same. In contrast, if $\mathcal{L}$ does not contain qualified number restrictions, then we introduce infinitely many copies d^i of any r-successor. The reason is that later we want to amalgamate the unravellings of $\mathcal{I}_1$ and $\mathcal{I}_2$, and, therefore, need the same number of r-successors satisfying the same $\mathcal{L}_S$-concepts in both unravellings. Construct $\mathcal{J}_2$ from $\mathcal{I}_2$ and the point e_1 in the same way.

We define an S-isomorphism ρ between $\mathcal{J}_1$ and $\mathcal{J}_2$ as the union of partial S-isomorphisms $\rho_0 \subseteq \rho_1 \subseteq \rho_2 \subseteq \ldots$, where

- ρ_n is an S-isomorphism between the restrictions of $\mathcal{J}_1$ and $\mathcal{J}_2$ to the points of *length not exceeding* n; i.e., points $(d_1, r_2, d_2^{i_2}, \ldots, d_m^{i_m}) \in \Delta^{\mathcal{J}_1}$, and $(e_1, r_2, e_2^{i_2}, \ldots, e_m^{i_m}) \in \Delta^{\mathcal{J}_2}$, where $m \leq n$;
- for each $w \in \mathbf{dom}(\rho_n)$: $w \sim_S \rho_n(w)$.

The sequence of partial S-isomorphisms is inductively defined as follows. For the induction base, set $\rho_1((d_1)) = (e_1)$. Consider the induction step. Suppose that ρ_n has been defined. Assume $\rho_n(w_1) = w_2$, where

$$w_1 = (d_1, r_2, d_2^{i_2}, \ldots, r_n, d_n^{i_n}), \text{ and}$$
$$w_2 = (e_1, s_2, e_2^{j_2}, \ldots, s_n, e_n^{j_n}).$$

Observe that $r_n \in S \cup S^-$ implies $s_n = r_n$ since ρ_n is a partial S-isomorphism. Now ρ_{n+1} is defined by adding, for each (w_1, w_2), the following pairs to ρ_n:

- If $r \in S \cup S^-$ with $r_n \neq r^-$ and $\mathcal{L}$ contains qualified number restrictions, we can take (by Point 4 above) an S-invariant bijection b between the $r^{\mathcal{I}_1}$-successors of d_n and the $r^{\mathcal{I}_2}$-successors of e_n and extend ρ_n with the set
$$\{((w_1, r, d^0), (w_2, r, b(d)^0) \mid (d_n, d) \in r^{\mathcal{I}_1}\}.$$
- If $r \in S \cup S^-$ with $r_n = r^-$ and $\mathcal{L}$-contains qualified number restrictions, consider the sets
$$B_1 = \{d \mid (d_n, d) \in r^{\mathcal{I}_1}\} \setminus \{d_{n-1}\} \text{ and } B_2 = \{e \mid (e_n, e) \in r^{\mathcal{I}_2}\} \setminus \{e_{n-1}\}.$$
We have $d_{n-1} \sim_S e_{n-1}$. Hence, by Point 4 above, there exists an S-invariant bijection b between B_1 and B_2. Extend ρ_n with the set
$$\{((w_1, r, d^0), (w_2, r, b(d)^0) \mid d \in B_1\}.$$
- If $r \in S \cup S^-$ and $\mathcal{L}$ does not contain qualified number restrictions, we find (by Points 2 and 3) a bijection b between the sets
$$B_1 = \bigcup_{i \geq 0} \{d^i \mid (d_n, d) \in r^{\mathcal{I}_1}\} \text{ and } B_2 = \bigcup_{i \geq 0} \{e^i \mid (e_n, e) \in r^{\mathcal{I}_2}\}$$
such that $b(d^i) = e^j$ implies $d \sim_S e$. Extend ρ_n with the set
$$\{((w_1, r, d), (w_2, r, b(d))) \mid d \in B_1\}.$$

- Finally, take an S-invariant bijection b between the remaining points $(w_1, r_{n+1}, d^i_{n+1})$ of $\Delta^{\mathcal{J}_1}$ and $(w_2, s_{n+1}, e^i_{n+1})$ of $\Delta^{\mathcal{J}_2}$. This is possible, because of Point 1 above and the introduction of the dummy relation symbol δ, which ensures that, for each $d \in \Delta^{\mathcal{I}_1}$, there are infinitely many $(w_1, \delta, d^i) \in \Delta^{\mathcal{J}_1}$ with $d \sim^{\mathcal{L}}_S (w_1, \delta, d^i)$, and, correspondingly, for $e \in \Delta^{\mathcal{J}_2}$. Add b to ρ_n.

The mapping $\rho = \bigcup_{n \geq 1} \rho_n$ is an S-isomorphism between $\mathcal{J}_1$ and $\mathcal{J}_2$. The required model $\mathcal{I}$ is now constructed by taking the model $\mathcal{J}_1$ but interpreting the $A \in \mathsf{N_C} \cap \mathsf{sig}(\Psi_2)$ as $\rho^{-1}(A^{\mathcal{J}_2})$ and the $r \in \mathsf{N_R} \cap \mathsf{sig}(\Psi_2)$ as $\rho^{-1}(r^{\mathcal{J}_2})$. □

Proposition 9. *Let $\mathcal{L}$ be any of the languages $\mathcal{ALC}$, $\mathcal{ALCQ}$, $\mathcal{ALCI}$, $\mathcal{ALCQI}$, $\mathcal{ALCU}$, $\mathcal{ALCQU}$, $\mathcal{ALCIU}$, $\mathcal{ALCQIU}$. Then $\mathcal{QL_L}$ has interpolation.*

Proof. Let $\mathcal{L}$ be any of the DLs listed in the proposition. Assume that there exists a set Ψ of $\mathcal{QL_L}$-sentences and $\mathcal{L}$-concepts C_0, D_0 with $\Psi \models C_0 \sqsubseteq D_0$ such that there does not exist an interpolant $I(\Psi, C_0 \sqsubseteq D_0)$. Let $S = \mathsf{sig}(\Psi) \cap \mathsf{sig}(C_0 \sqsubseteq D_0)$ and

$$\Psi_S = \{C \sqsubseteq D \mid \Psi \models C \sqsubseteq D, \mathsf{sig}(C \sqsubseteq D) \subseteq S\}.$$

Then $\Psi \models \Psi_S$ and $\Psi_S \not\models C_0 \sqsubseteq D_0$, by compactness. Take a model $\mathcal{I}_2'$ of Ψ_S with $e_1 \in (C_0 \sqcap \neg D_0)^{\mathcal{I}_2'}$. Take a model $\mathcal{I}_1'$ of Ψ containing a point d_1 such that $(\mathcal{I}_1', d_1) \sim^{\mathcal{L}}_S (\mathcal{I}_2', e_1)$. The existence of such a model follows again by compactness. Now the proof splits into two parts.

(i) Assume first that $\mathcal{L}$ contains the universal role. Then, for all $\mathcal{L}_S$-concepts C, $C^{\mathcal{I}_2'} = \emptyset$ iff $C^{\mathcal{I}_1'} = \emptyset$, because $d_1 \in (\exists u.C)^{\mathcal{I}_1'}$ iff $e_1 \in (\exists u.C)^{\mathcal{I}_2'}$. It follows that, for $\Psi_1 = \Psi$ and $\Psi_2 = \emptyset$, the models $\mathcal{I}_1'$ and $\mathcal{I}_2'$ and points d_1 and e_1 satisfy the conditions of Proposition 8. We obtain a model $\mathcal{I}$ of Ψ such that $(C_0 \sqcap \neg D_0)^{\mathcal{I}} \neq \emptyset$. Hence $\Psi \not\models C_0 \sqsubseteq D_0$ and we have derived a contradiction.

(ii) Assume now that $\mathcal{L}$ does no contain the universal role. Let

$$\mathcal{C} = \{C \mid \ C \text{ a } \mathcal{L}_S\text{-concept with } \Psi \not\models C \sqsubseteq \bot\}$$

and take for each $C \in \mathcal{C}$ a model $\mathcal{I}_C$ of Ψ such that $C^{\mathcal{I}_C} \neq \emptyset$. Let $\mathcal{I}_1$ be the disjoint union of the models $\mathcal{I}_1'$ and $\mathcal{I}_C$, $C \in \mathcal{C}$. Correspondingly, let $\mathcal{I}_2$ be the disjoint union of the models $\mathcal{I}_2'$ and $\mathcal{I}_C$, $C \in \mathcal{C}$. Again, $\mathcal{I}_1, \mathcal{I}_2, d_1, e_1$ satisfy the conditions of Proposition 8 for $\Psi_1 = \Psi$ and $\Psi_2 = \emptyset$. Hence there exists a model $\mathcal{I}$ of Ψ with $(C_0 \sqcap \neg D_0)^{\mathcal{I}} \neq \emptyset$ and we have derived a contradiction. □

Proposition 10. *Let $\mathcal{L}$ be any of the languages $\mathcal{ALC}$, $\mathcal{ALCQ}$, $\mathcal{ALCI}$, $\mathcal{ALCQI}$. Then $(\mathcal{L}, \mathcal{QL_L})$ is robust under joins.*

Proof. Fix a DL $\mathcal{L}$ from the proposition. Let $\mathcal{O}_1$ and $\mathcal{O}_2$ be $\mathcal{L}$-ontologies. Assume $\mathcal{O}_1$ and $\mathcal{O}_2$ are S-inseparable w.r.t. $\mathcal{QL_L}$ with $\mathsf{sig}(\mathcal{O}_1) \cap \mathsf{sig}(\mathcal{O}_2) \subseteq S$. Let, for $i = 1, 2$,

$$\mathcal{C}_i = \{C \mid C \text{ a } \mathcal{L}_S\text{-concept with } \mathcal{O}_i \not\models C \sqsubseteq \bot\}.$$

By S-inseparability, $\mathcal{C}_1 = \mathcal{C}_2$. Take, for each $C \in \mathcal{C}_1$, a model $\mathcal{I}_C^1$ of $\mathcal{O}_1$ such that $C^{\mathcal{I}_C^1} \neq \emptyset$ and let $\mathcal{I}_1$ be the disjoint union of the models $\mathcal{I}_C^1$, $C \in \mathcal{C}_1$. Similarly, take, for each $C \in \mathcal{C}_1$, a model $\mathcal{I}_C^2$ of $\mathcal{O}_2$ such that $C^{\mathcal{I}_C^2} \neq \emptyset$ and let $\mathcal{I}_2$ be the disjoint union of the models $\mathcal{I}_C^2$, $C \in \mathcal{C}_1$. Then $\mathcal{I}_1$ and $\mathcal{I}_2$ satisfy the conditions of Proposition 8 for $\Psi_1 = \mathcal{O}_1$ and $\Psi_2 = \mathcal{O}_2$. Hence, there exists a model $\mathcal{I}$ of $\mathcal{O}_1 \cup \mathcal{O}_2$ such that $C^{\mathcal{I}} \neq \emptyset$ whenever $C \in \mathcal{C}_1$. It follows that $\mathcal{O}_1 \cup \mathcal{O}_2$ and $\mathcal{O}_i$ are S-inseparable w.r.t. $\mathcal{QL}_{\mathcal{L}}$, $i = 1, 2$. $\square$

3

Criteria and Evaluation for Ontology Modularization Techniques

Mathieu d'Aquin[1], Anne Schlicht[2],
Heiner Stuckenschmidt[2], and Marta Sabou[1]

[1] Knowledge Media Institute (KMi)
The Open University, Milton Keynes, UK
{m.daquin,r.m.sabou}@open.ac.uk
[2] University of Mannheim, Germany
{anne,heiner}@informatik.uni-mannheim.de

Summary. While many authors have argued for the benefits of applying principles of modularization to ontologies, there is not yet a common understanding of how modules are defined and what properties they should have. In the previous section, this question was addressed from a purely logical point of view. In this chapter, we take a broader view on possible criteria that can be used to determine the quality of a modules. Such criteria include logic-based, but also structural and application-dependent criteria, sometimes borrowing from related fields such as software engineering. We give an overview of possible criteria and identify a lack of application-dependent quality measures. We further report some modularization experiments and discuss the role of quality criteria and evaluation in the context of these experiments.

3.1 Introduction

Problems with large monolithical ontologies in terms of reusability, scalability and maintenance have lead to an increasing interest in techniques for extracting modules from ontologies. Currently, existing work suffers from the fact that the notion of modularization is not as well understood in the context of ontologies as it is in software engineering. While there is a clear need for ontology modularization, there are no well-defined and broadly accepted ideas about the criteria that define a good module. As a result, several approaches have been recently used to extract modules from ontologies, each of them implementing its own intuition about what a module should contain and what should be its qualities. In addition, a number of formal and informal modularization criteria have been proposed that are strongly influenced by certain use cases for modularization. This lack of consensus about quality criteria hinders the development of the field as a whole. On one hand, it is difficult to take up the results of the field outside itself because of a lack of guidelines about which technique to choose under which circumstances. On the other hand, within the field, it is impossible to compare the various techniques to each other.

Our hypothesis is that there is no universal way to modularize an ontology and that the choice of a particular technique or approach should be guided by the requirements of the application or scenario relying on modularization. In fact, we have already observed a strong correspondence between the two major use cases in which modularization is needed and the two types of techniques that are used [7]. First, we distinguish scenarios where a large, monolithic ontology needs to be split up in order to allow its easier *maintenance and use* (e.g., by using reasoners and visualization tools). Accordingly, a significant group of techniques reported in the literature perform *ontology partitioning* by dividing an ontology into a set of significant modules that together form the original ontology [11, 6, 15]. The second class of scenarios, geared towards *selective use and reuse*, are those where a smaller part of an ontology that covers certain aspects of the domain is identified as a basis for a specific application. Candidate parts for reuse need to be small enough to be easily visualized and integrated in other applications than the one they have been initially built for. *Module extraction* techniques address such scenarios and refer to extracting a module from an ontology to be used for a particular purpose, i.e. covering a particular subvocabulary of the original ontology [14, 12, 7].

Based on the observation above, we believe that modularization criteria should be defined in terms of the applications for which the modules are created. In the remainder of this chapter, we survey existing criteria that can be used to evaluate modularization. Our goal is to provide a framework for evaluating and comparing modularization techniques according to application requirements, and so, a guideline to chose the right technique or combination of technique in a given scenario. Accordingly, we describe a set of experiments in which we apply a number of modularization techniques and analyze the results regarding the considered criteria. Also, as our main hypothesis is that the evaluation of modularization depends on the application requirements, these experiments are based a concrete applications scenario: the selection of relevant knowledge in existing ontology to be used in annotation.

The goal is to characterize the requirements of this particular application using the reviewed criteria and thus, to analyze the results of existing ontology modularization techniques regarding these requirements. Looking at the results of these experiments, we aim at better understanding the fundamental assumptions underlying the current modularization techniques and thus, at providing the building blocks for a more comprehensive evaluation, helping the application developers in choosing the appropriate technique and guiding the designers of techniques in further developments.

3.2 Use Cases for Modularization

The increasing awareness of the benefits of ontologies for information processing in open and weakly structured environments has lead to the creation of a number of such ontologies for real world domains. In complex domains such as medicine these ontologies can contain thousands of concepts. Examples of such large ontologies are the NCI Thesaurus with about 27.500 and the Gene Ontology with about 22.000 concepts. Other examples can be found in the area of e-commerce where product classification such as the UNSPSC or the NAICS contain thousands of product categories. While being useful for many applications, the size of these ontologies causes new problems that affect different steps of the ontology life cycle.

Maintenance

Ontologies that contain thousands of concepts cannot be created and maintained by a single person. The broad coverage of such large ontologies normally requires a team of experts. In many cases these experts will be located in different organizations and will work on the same ontology in parallel. An example for such a situation is the Gene Ontology that is maintained by a consortium of experts.

Publication

Large ontologies are mostly created to provide a standard model of a domain to be used by developers of individual solutions within that domain. While existing large ontologies often cover a complete domain, the providers of individual solutions are often only interested in a specific part of the overall domain. The UNSPSC classification for example contains categories for all kinds of products and services while the developers of an online computer shop will only be interested in those categories related to computer hardware and software.

Validation

The nature of ontologies as reference models for a domain require a high degree of quality of the respective model. Representing a consensus model, it is also important to have proposed models validated by different experts. In the case of large ontologies it is often difficult, if not impossible, to understand the model as a whole due to cognitive limits. What is missing is an abstracted view on the overall model and its structure as well as the possibility to focus the inspection of a specific aspect.

Processing

On a technical level, very large ontologies cause serious scalability problems. The complexity of reasoning about ontologies is well known to be critical even for smaller ontologies. In the presence of ontologies like the NCI Thesaurus, not only reasoning engines but also modelling and visualization tools reach their limits. Currently, there is no modelling tool that can provide convenient modelling support for ontologies of the size of the NCI ontology.

All these problems are a result of the fact that a large ontology is treated as a single monolithic model. Most problems would disappear, if the overall model consists of a set of coherent modules about a certain subtopic that can be used independently of the other modules while still containing information about its relation to these other modules.

3.3 Modularization Approaches

We consider an ontology O as a set of axioms (subclass, equivalence, instantiation, etc.) and the signature $Sig(O)$ of an ontology O as the set of entity names occurring in the axioms of O, i.e. its vocabulary.

In the following, we deal with several approaches for ontology modularization, having different assumptions about the definition of an ontology module. The assumption we adopt as a basis for our discussion is that a module is considered to be a significant and self-contained sub-part of an ontology. Therefore, an module $M_i(O)$ of an ontology O is also a set of axioms (an ontology), with the minimal constraint that $Sig(M_i(O)) \subseteq Sig(O)$. Note that, while it may often be desirable, it is not always the case that $M_i(O) \subseteq O$.

3.3.1 Ontology Partitioning Approaches

The task of partitioning an ontology is the process of splitting up the set of axioms into a set of modules $\{M_1, \cdots, M_k\}$ such that each M_i is an ontology and the union of all modules is semantically equivalent to the original ontology O. Note that some approaches being labeled as *partitioning* methods do not actually create *partitions*, as the resulting modules may overlap. There are several approaches for ontology partitioning that have been developed for different purposes.

The approach of [11] aims at improving the efficiency of inference algorithms by localizing reasoning. For this purpose, this technique minimizes the shared language (i.e. the intersection of the signatures) of pairs of modules. A message passing algorithm for reasoning over the distributed ontology is proposed for implementing resolution-based inference in the separate modules. Completeness and correctness of some resolution strategies is preserved and others trade completeness for efficiency.

The approach of [6] partitions an ontology into a set of modules connected by $\mathcal{E}$-Connections. This approach aims at preserving the completeness of local reasoning within all created modules. This requirement is supposed to make the approach suitable for supporting selective use and reuse since every module can be exploited independently of the others.

A tool that produces sparsely connected modules of reduced size was presented in [15]. The goal of this approach is to support maintenance and use of very large ontologies by providing the possibility to individually inspect smaller parts of the ontology. The algorithm operates with a number of parameters that can be used to tune the result to the requirements of a given application.

3.3.2 Module Extraction Approaches

The task of module extraction consists in reducing an ontology to the sub-part, the module, that covers a particular sub-vocabulary. This task has been called segmentation in [14] and traversal view extraction in [12]. More precisely, given an ontology O and a set $SV \subseteq Sig(O)$ of terms from the ontology, a module extraction mechanism returns a module M_{SV}, supposed to be the relevant part of O that covers the sub-vocabulary SV ($Sig(M_{SV}) \supseteq SV$). Techniques for module extraction often rely on the so-called *traversal approach*: starting from the elements of the input sub-vocabulary, relations in the ontology are recursively "traversed" to gather relevant (i.e. related) elements to be included in the module.

Such a technique has been integrated in the PROMPT tool [12], to be used in the Protégé environment. This approach recursively follows the properties around a selected class of the ontology, until a given distance is reached. The user can exclude certain properties in order to adapt the result to the needs of the application.

The mechanism presented in [14] starts from a set of classes of the input ontology and extracts related elements on the basis of class subsumption and OWL restrictions. Some optional filters can also be activated to reduce the size of the resulting module. This technique has been implemented to be used in the Galen project and relies on the Galen Upper Ontology.

Inspired from the previously described techniques, [7] defines an approach for the purpose of the dynamic selection of relevant modules from online ontologies. The input sub-vocabulary can contain either classes, properties, or individuals. The mechanism is fully automatized, is designed to work with different kinds of ontologies (from simple taxonomies to rich and complex OWL ontologies) and relies on inferences during the modularization process.

3.4 Evaluation Criteria for Modularization

In the previous section, we have briefly presented a number of different approaches for ontology partitioning and module extraction. In this section, we take a closer look at different criteria for evaluating either the modularization resulting from the application of a modularization technique or the system implementing the modularization technique. Before that, we start by looking at criteria that have been adopted for the classical notion of a module in software engineering.

3.4.1 Criteria from Software Engineering

The author of [9] reviews a set of features of software engineering modules with respect to what is called in this chapter the *requirements for logical modules*. From this analysis, two general criteria characterizing software engineering modules can be considered: encapsulation and independence.

Encapsulation

Encapsulation refers to the distinction between the interface and the body (or implementation) of a program in software engineering. This distinction does not really apply when using Semantic Web technologies, but can be related to other notions like substitutability and information hiding. Indeed, the fact that a module can be easily exchanged for another, or internally modified, without side-effects on the application can be a good indication of the quality of the module. Module extraction techniques are intrinsically related to information hiding, since they aim at removing from the ontology the elements that are not related to the given sub-vocabulary, playing the role of an interface between the ontology and the application.

Independence

A well-designed software module should be independent from the other modules used in the same applications in the sense that it should be reusable and exploitable separately. The same applies in ontology engineering, where ontology modules have to be self-contained and reusable components. Additionally, having independent modules is a way to improve the scalability of reasoning mechanisms by allowing to focus on a smaller set of elements (in the case of module extraction techniques, see e.g. [14]), or the use of distributed reasoning (in the case of partitioning techniques, see e.g. [11, 6]).

3.4.2 Evaluating Modularizations

Logical Criteria

If we look at ontologies as logical theories, it is a natural approach to define modularization criteria in terms of their logical properties, i.e. looking at their entailments. Recently, several papers have been interested in defining such criteria.

Local Correctness

is probably the most obvious formal relation between a module $M_i(O)$ and its original ontology O. It states that every axiom being entailed by $M_i(O)$ should also be entailed by O. In other terms, nothing has been added in the module that was not originally in the ontology.

Local Completeness

is the reverse property of local correctness. It is considered in many studies as the most important logical criterion concerning modularization. A module is said to be locally complete w.r.t. a local signature $Loc(M_i) \subseteq Sig(M_i)$ (e.g., the original set of entities of interest in an extraction technique) if every entailment of O that concerns only elements of $Loc(M_i)$ is preserved in $M_i(O)$. Local completeness has been formalized for example in [3] using the notion of *conservative extension* and also relates to the one of *uniform interpolant* described in [6].

Structural Criteria

[13] describes a set of criteria that can be computed on the basis of the structure of the modularized ontology. The criteria are inherently related to the previously mentioned software engineering criteria as they have been designed to trade-off maintainability as well as efficiency of reasoning in a distributed system, using distributed modules.

Size

Despite its evident simplicity, the relative size of a module (number of classes, properties and individuals) compared to its source ontology is among the most important indicators of the efficiency of a modularization technique. Indeed, the size of a module has a strong influence on its maintainability and on the robustness of the applications relying on it: a big amount of information in one module leads to less flexibility in its exploitation and evolution. On the other hand, too small modules would not cover a sufficiently broad domain to be useful and would lead to problems related to other criteria (e.g. connectedness).

Intra-Module Distance

It is worth to measure how the terms described in a module *move closer to each other* compared to the original ontology, as an indication of the simplification of the structure of the module. This *intra-module distance* is computed by counting the number of relations in the shortest path from one entity to the other. This is particularly relevant in the case of module extraction techniques, where reducing the distance between the input terms facilitates their joint visualization and helps in understanding their relationship in the original ontology.

Quality of the Modules

There are two different kinds of approaches for determining the quality of an existing ontology that can be used to evaluate the quality of ontology modules. While some approaches analyze the representational structures of an ontology on the basis of the actual content, others compare the content of the ontology with alternative representations of the domain (e.g. a document corpus) to determine how well it models relevant aspects of the domain it describes. We propose to use and combine these approaches in order to get an estimation of the suitability of a module for describing a certain aspect of a domain.

Module cohesion

Cohesion denotes the degree of relatedness of elements within the same module. In the area of software engineering, a number of measures of cohesion have been defined that try to measure the connectedness of methods in a module based on criteria such as shared instance variables [2]. For the case of ontologies, Yao and his colleagues propose a set of cohesion metrics based on the structure of the inheritance hierarchy [17] and show that these metrics correlate with the intuition of human evaluators: the *number of root classes* in the hierarchy, the *number of leaf classes*, and the *maximum depth* of the hierarchy. The relevance of the definitions depends on all of these factors and only their combination provides a suitable indicator.

Richness of the representation

Another important aspect related to the quality of modules is the amount of conceptual information retained in the module. We can measure this amount using some criteria that are inspired by the notion of schema richness proposed by Tatir and his colleagues [16] to measure the quality of ontologies. Here measuring the richness of the specifications in a module is based on the amount of relational information present in relation to the number of classes. We distinguish between the richness of the subsumption hierarchy – the average number of subclass relations per class – and the richness of the relations between classes – the average number of domain relations per class. It is clear that the richness of semantic information in a module strongly depends on the richness of the ontology the module originated from. A better indication of the quality is therefore provided by comparing the richness of the module with the richness of the corresponding set of concepts in the original ontology.

Domain coverage

In the context of real applications domain coverage is the most important criterion as it determines how well the module fits the representational requirements of the application at hand. In order to be able to determine the domain coverage, we need a suitable representation of the domain that should be covered by the module. In cases where no instance data exists, we can adopt techniques of data-driven ontology evaluation that have been proposed in the area of ontology learning [1]. The idea is to compare the ontology with a corpus of documents in order to determine how well the ontology describes the content of the documents. These evaluations help in determining if the modularization technique have generated *significant* module according to the different domains or topics covered by the original ontology. What have to be evaluated is whether or not these modules are actually focused on a restricted number of domains, and if the domains are localized in the modularization, i.e. if they have not been spread over an important number of modules.

Relations between Modules

Connectedness

The independence (see Section 3.4.1), and so the efficiency, of a set of modules resulting from a partitioning technique can be estimated by looking at the degree of interconnectedness of the generated modules. A modularized ontology can be depicted as a graph, where the axioms are nodes and edges connect every two axioms that share a symbol. The connectedness of a module is then evaluated on the basis of the number of edges it shares with other modules.

Redundancy

In case of partitioning techniques that allow modules to overlap, redundancy is a common way of improving efficiency and robustness. On the other hand, having to deal with redundant information increases the maintenance effort, and it has been shown in [11] that reasoning on non-redundant representations of parts of the complete model can lead to performance improvements.

In addition, computing the level of redundancy (the overlap) in modules generated using different techniques can be a way to compare and relate these techniques. More precisely, it can indicate whether these techniques rely on similar intuitions, if one is more general than the other, or, on the contrary, if they result in very different (and possibly complementary) modules.

Inter-module distance

Counting the number of modules that have to be considered to relate two entities can help in characterizing the communication effort caused by the partition of an ontology. Indeed, if the modules resulting from an ontology partitioning technique are intended to be used on different machines, over a network, the *inter-module distance* gives an indication of the amount of distant access to other modules that is required to manipulate the considered entities.

3.4.3 Application Criteria

In [7] the authors focus on the use of modularization for a particular application. This leads to the definition of several criteria, most of them characterizing the adequacy of the design of a modularization tool with respect to constraints introduced by the application: assumptions on the ontology, the level of user interaction, and the availability of tools for using the resulting modules. Additionally, applications may have different requirements regarding the performance of the modularization tool.

Assumptions on the ontology

Most of the existing approaches rely on some assumptions. For example, those described in [5] and [14] are explicitly made to work on OWL ontologies, whereas [15] can be used either on RDF or OWL but only exploits RDF features. In [14], the ontology is required to be well-designed and to use complex OWL constructs to describe classes. Moreover, some of the filters used to reduce the size of a module are dependent on elements of the Galen upper ontology.

Level of user interaction

In many systems the required user entries are limited to the inputs of the algorithm. In certain cases, some numerical parameters can be required [15] or some additional procedures can be manually (de)activated [14]. The technique in [12] has been integrated in the Protégé ontology editor to support knowledge reuse during the building of a new ontology. In this case, modularization is an interactive process where the user has the possibility to extend the current module by choosing a new starting point for the traversal algorithm among the *boundary classes* of the module.

Use of modules

Regarding the aspect of actually using modules in other applications, we only know of two approaches that make their modules available to reasoners/theorem provers (but not to any other applications). The modules extracted in [5] are linked together using $\mathcal{E}$-Connections and aim at being used in a reasoner. In a similar way, the knowledge base partitions created by the approache of [11] are used in a dedicated theorem prover.

Performance

Most of the papers concerning modularization techniques do not give any indication about the performance of the employed method (with the noticeable exception of [14]). Performance is a particularly important element to be considered when using a modularization technique for the purpose of an application. Different applications may have different requirements, depending on whether the modularization is intended to be used dynamically, at run-time, or as a "batch" process.

3.5 Experiments with Modularization Techniques and Criteria

In this section, we apply the criteria described in the previous section to a particular application scenario, on the basis of two well defined examples. The idea is to evaluate how these criteria can be used in characterizing the application requirements and the assumptions underlying the modularization techniques.

3.5.1 The Knowledge Selection Scenario

Knowledge selection has been described in [7] as the process of selecting the relevant knowledge components from online available ontologies and has been in particular applied to the Magpie application. Magpie [8] is a Semantic Web browser which helps users to quickly make sense of the information provided by a Web page by allowing them to visually access the semantic data associated with that page. Available as a browser plugin in which a set of classes are displayed, Magpie can identify instances of these classes in the current Web page and highlight them with the color associated to each class. Core to Magpie is a manually selected ontology that contains the information needed to identify the relevant instances in Web pages.

In our current work we are extending Magpie towards open semantic browsing in which the tool automatically selects and combines online ontologies relevant to the content of the current page. As such, the user is relieved from manually choosing a suitable ontology every time he wishes to browse new content. Such an extension of our tool relies on mechanisms that can not only dynamically select appropriate ontologies from the Web, but can also extract from these ontologies the relevant and useful parts to describe classes in the current Web page.

Our previous work and experiences in ontology selection [10] made it clear that modularization may play a crucial role in complementing the current selection techniques. Indeed, selection algorithms tend to run into two major problems. First, if the selection returns a large ontology this is virtually useless for a tool such as Magpie which only visualises a relatively small number of classes at a time. Unfortunately, in the experiments we have performed large ontologies are often returned. What is needed instead is that the selection process returns a part (module) of the ontology that defines the relevant set of terms. A second problem is that in many cases it is difficult to find a single ontology that covers all terms (we observed this knowledge sparseness phenomenon in [10]). However, a combination of one or more ontologies could cover all the query terms. This problem is related to modularization in the sense that it is easier to combine small and focused knowledge modules than ontologies of large size and coverage.

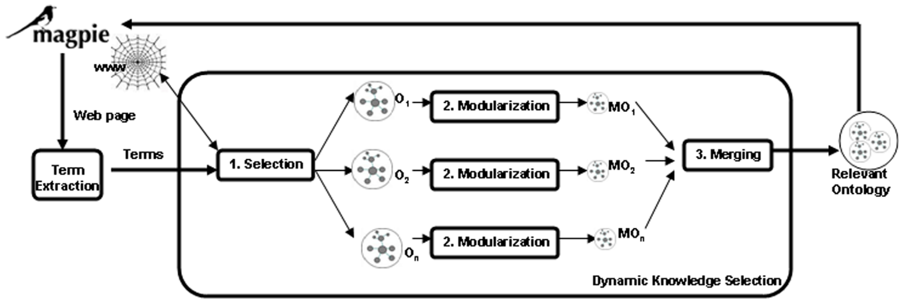


Fig. 3.1. The knowledge selection process and its use for semantic browsing with Magpie

These considerations justify the need to extend selection techniques with modularization capabilities. In Figure 3.1 we depict the three major generic steps of the *knowledge selection* process that integrates ontology selection, modularization and merging.

3.5.2 Experimental Setting

In the scenario described in the previous section, modularization is integrated in a fully automatic process, manipulating automatically selected online ontologies for the purpose of annotation in Magpie. In this section, we simulate the process of knowledge selection on two examples, using four different techniques, in order to evaluate and compare their results[1]. The purpose is to characterize the requirements of this particular scenario using the criteria defined in Section 3.4, and to show how modularization techniques respond to the selected experiments regarding these requirements.

[1] Actual results are available at
`http://webrum.uni-mannheim.de/math/lski/Modularization`

The Examples

We consider two examples, originally described in the context of ontology selection in [10], where the goal is to obtain an ontology module for the annotation of news stories. We simulate the scenario described in Section 3.5.1 by manually extracting relevant keywords in these stories, using ontology selection tools[2] to retrieve ontologies covering these terms, and then applying modularization techniques on these ontologies (steps 1 and 2 in figure 3.1).

In the first example, we consider the case where we want to annotate the news stories available on the KMi website[3]. We used the keywords *Student*, *Researcher*, and *University* to select ontologies to be modularized, and obtain three ontologies covering these terms:

ISWC: `http://annotation.semanticweb.org/iswc/iswc.owl`
KA: `http://protege.stanford.edu/plugins/owl/owl-library/ka.owl`
Portal: `http://www.aktors.org/ontology/portal`

It is worth to mention that this example is designed to be simple: we have chosen a well covered domain and obtained three well defined OWL ontologies of small size (33 to 169 classes).

The second example was used in [10] to illustrate the difficulties encountered by ontology selection algorithms. Consequently, it also introduces more difficulties for the modularization techniques, in particular because of the variety of the retrieved ontologies in terms of size and quality. It is based on the following news snippet:

"The Queen will be 80 on 21 April and she is celebrating her birthday with a family dinner hosted by Prince Charles at Windsor Castle"[4]

Using the keywords *Queen*, *Birthday* and *Dinner*, we obtained the following ontologies, covering (sometimes only partially) this set of terms:

OntoSem: `http://morpheus.cs.umbc.edu/aks1/ontosem.owl`
TAP: `http://athena.ics.forth.gr:9090/RDF/VRP/Examples/tap.rdf`
Mid-Level: `http://reliant.teknowledge.com/DAML/Mid-level-ontology.owl`, covering only the terms *Queen* and *Birthday*

Compared to Example 1, the ontologies used in Example 2 are bigger (from 1835 classes in **Mid-Level** to 7596 in **OntoSem**). Moreover, they contain different levels of descriptions. For example, **OntoSem** is a big, complex OWL ontology containing a lot of properties (about 600), whereas TAP is simple RDFS taxonomy without any properties. In that sense, we use Example 1 to assess basic characteristics of the modularization techniques and then, rely on Example 2 to show how these characteristics are influenced by the properties of the ontologies.

[2] In particular Watson (`http://watson.kmi.open.ac.uk`).
[3] `http://news.kmi.open.ac.uk/`
[4] `http://news.billinge.com/1/hi/entertainment/4820796.stm`

Evaluated criteria

Logical criteria are of particular importance when the modules resulting of the modularization techniques are intended to be used in reasoning mechanisms, but should not be emphasized in our use case, which focuses on a human interpretation of the module.

On the contrary, structural criteria are fundamental indicators for estimating the efficiency of the modularization techniques. Indeed, the size of the returned ontology module is crucial since Magpie only needs relevant parts of ontologies, small enough to be visualized within the browser and to be easily interpreted by the user in relation with the current Web page. However, it is essential to keep enough knowledge in the module to maintain a understandable structure around the considered terms.

Evaluating the quality of the resulting modules is a crucial task, in particular in applications where modularization is intended to facilitate ontology reuse. However, applying and interpreting the metrics for evaluating module quality require important (human) effort, which go beyond the scope of our experiments, intending to simulate an automatic process.

Criteria dedicated to sets of interconnected modules resulting from partitioning techniques – *redundancy*, *connectedness*, and *inter-module distance* – are not relevant in the considered scenario, as we are only interested in one module. One of the goals of modularization in our use case is to facilitate the exploitation, and so the interpretation, of the knowledge related to the input terms in ontologies. In that sense, the *intra-module distance* between these terms should be reduced in such a way that they can be considered and apprehended together by the user.

In the knowledge selection scenario, modularization is integrated in a complete process, leading to particular constraints concerning application criteria. The results of Example 2 should help to better understand which *assumptions* the different techniques rely on. Since knowledge selection is fully automatized, the required level of *user interaction* should be minimal. Finally, in the considered scenario, modularization is intended to be used at run-time, leading to fundamental constraints concerning the *performance* of the modularization tool.

3.5.3 Experimented Techniques

As already described in [7], it is quite obvious that module extraction techniques fit better in the considered scenario than partitioning tools. Indeed, we want to obtain *one* module covering the set of keywords used for the selection of the ontology and constituting a *sub-vocabulary* of this ontology. However, the result of partitioning techniques can also be used by selecting the set of generated modules that cover the considered terms. The criteria are then evaluated on this set of modules as grouped together by union.

We have chosen to consider only available partitioning and module extraction techniques that are sufficiently stable and that can easily be used on the previously described ontologies (e.g., without requiring language conversion). We have selected two ontology partitioning tools:

PATO: A standalone application described in [15].

SWOOP: The partitioning functionality included in the SWOOP ontology editor and described in [5].

and two module extraction tools:

KMi: A standalone application developed at KMi for the purpose of the knowledge selection scenario, as described in [7].

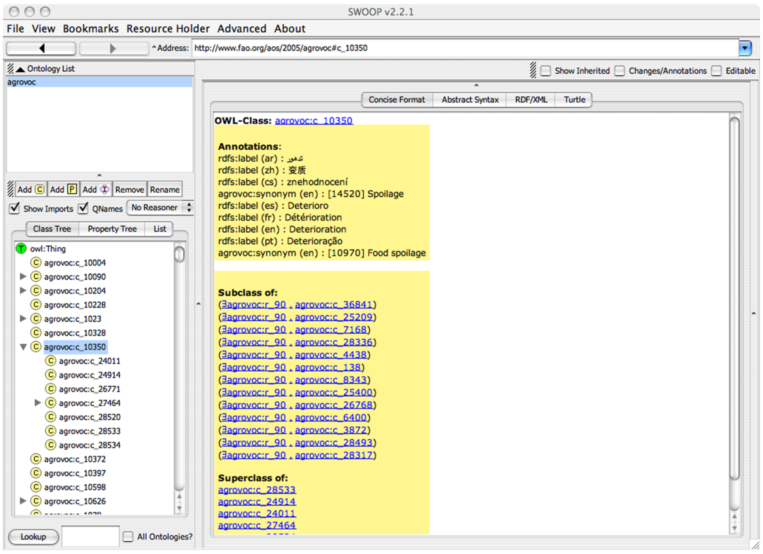
Prompt: The module extraction feature of the Prompt toolkit, integrated as a plugin of the Protégé ontology editor, as described in [12].

These tools are described in more details below.

Ontology Partitioning Technique: SWOOP

SWOOP[5] is a popular ontology editor, focused on OWL ontologies and relying on an hypertext-like way of navigating in the ontology entities (see Figure 3.2). SWOOP is developed by the same group who made the Pellet OWL reasoner, and inference capabilities can be used during the editing process thanks to Pellet. Another particularity of SWOOP is that, in addition to standard OWL ontologies, it can manage $\mathcal{E}$-Connections based ontologies: local ontologies linked together by $\mathcal{E}$-Connections [4].

SWOOP integrates a fully automatic ontology partitioning functionality. This functionality, based on the paper [5], is supposed to divide standard OWL ontologies to create a set of local ontologies, linked together by $\mathcal{E}$-Connections. This partitioning feature of the SWOOP editor is further referred to as **SWOOP**.



owl:Thing
- agrovoc:c_10004
- agrovoc:c_10090
- agrovoc:c_10204
- agrovoc:c_10228
- agrovoc:c_1023
- agrovoc:c_10328
- agrovoc:c_10350
  - agrovoc:c_24011
  - agrovoc:c_24914
  - agrovoc:c_26771
  - agrovoc:c_27464
  - agrovoc:c_28520
  - agrovoc:c_28533
  - agrovoc:c_28534
- agrovoc:c_10372
- agrovoc:c_10397
- agrovoc:c_10598
- agrovoc:c_10626

OWL-Class: agrovoc:c_10350

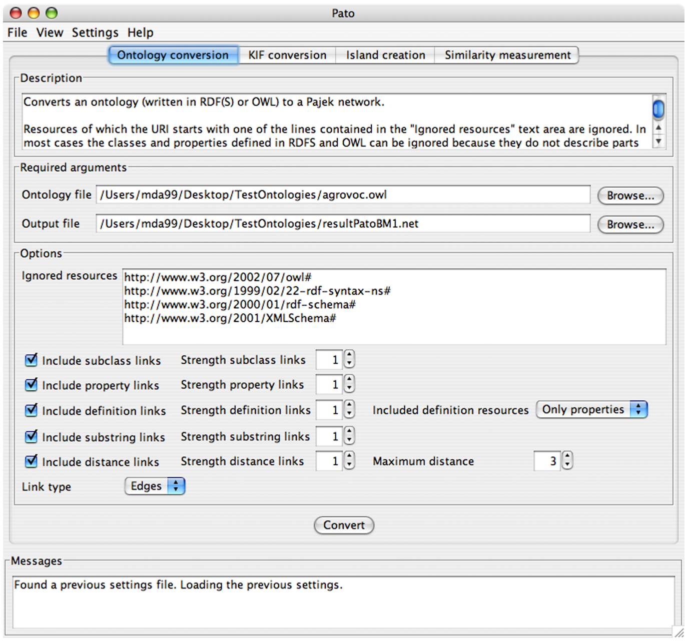
Annotations:
rdfs:label (ar) : تدهور
rdfs:label (zh) : 变质
rdfs:label (cs) : znehodnocení
agrovoc:synonym (en) : [14520] Spoilage
rdfs:label (es) : Deterioro
rdfs:label (fr) : Détérioration
rdfs:label (en) : Deterioration
rdfs:label (pt) : Deterioração
agrovoc:synonym (en) : [10970] Food spoilage

Subclass of:
(∃agrovoc:r_90 . agrovoc:c_36841)
(∃agrovoc:r_90 . agrovoc:c_25209)
(∃agrovoc:r_90 . agrovoc:c_7168)
(∃agrovoc:r_90 . agrovoc:c_28336)
(∃agrovoc:r_90 . agrovoc:c_4438)
(∃agrovoc:r_90 . agrovoc:c_138)
(∃agrovoc:r_90 . agrovoc:c_8343)
(∃agrovoc:r_90 . agrovoc:c_25400)
(∃agrovoc:r_90 . agrovoc:c_26768)
(∃agrovoc:r_90 . agrovoc:c_6400)
(∃agrovoc:r_90 . agrovoc:c_3872)
(∃agrovoc:r_90 . agrovoc:c_28493)
(∃agrovoc:r_90 . agrovoc:c_28317)

Superclass of:
agrovoc:c_28533
agrovoc:c_24914
agrovoc:c_24011
agrovoc:c_27464

Fig. 3.2. Screenshot of the SWOOP editor

[5] `http://www.mindswap.org/2004/SWOOP/`



Description

Converts an ontology (written in RDF(S) or OWL) to a Pajek network.

Resources of which the URI starts with one of the lines contained in the "Ignored resources" text area are ignored. In most cases the classes and properties defined in RDFS and OWL can be ignored because they do not describe parts

Required arguments

Ontology file /Users/mda99/Desktop/TestOntologies/agrovoc.owl

Output file /Users/mda99/Desktop/TestOntologies/resultPatoBM1.net

Options

Ignored resources
```
http://www.w3.org/2002/07/owl#
http://www.w3.org/1999/02/22-rdf-syntax-ns#
http://www.w3.org/2000/01/rdf-schema#
http://www.w3.org/2001/XMLSchema#
```

✓ Include subclass links	Strength subclass links 1		
✓ Include property links	Strength property links 1		
✓ Include definition links	Strength definition links 1	Included definition resources	Only properties
✓ Include substring links	Strength substring links 1		
✓ Include distance links	Strength distance links 1	Maximum distance	3
Link type	Edges		

Messages

Found a previous settings file. Loading the previous settings.

Fig. 3.3. Screenshot of the **PATO** tool

From these elements we can already get an evaluation of some of the *application criteria*: **SWOOP** is fully automatic, works only on OWL ontologies (without imports) and there exist tools (the SWOOP editor, Pellet) to exploit the resulting set of modules.

Ontology Partitioning Technique: PATO

PATO is a standalone application written in java (see Figure 3.3) and implementing the technique described in [15]. In principle, **PATO** divides an ontology into a set of modules by first computing a *dependency graph* between the entities of the ontology, relying on RDF(S) relations between them. Entities are then clustered according to measures inspired from the field of network analysis, aiming at minimizing the interconnections between modules.

Concerning the application criteria, as it can be seen in Figure 3.3, **PATO** is supposed to be fine-tuned using different parameters at each step of the partitioning process. However, an ongoing work is currently conducted towards the automatic configuration of **PATO** [13]. No particular reasoner is available for exploiting modules resulting from this tools. However, each module is a self-contained OWL ontology that can be used with usual tools. Finally, even if it works with most of the ontologies in RDF(S) or OWL, **PATO** only exploits the representation primitive of RDF(S).

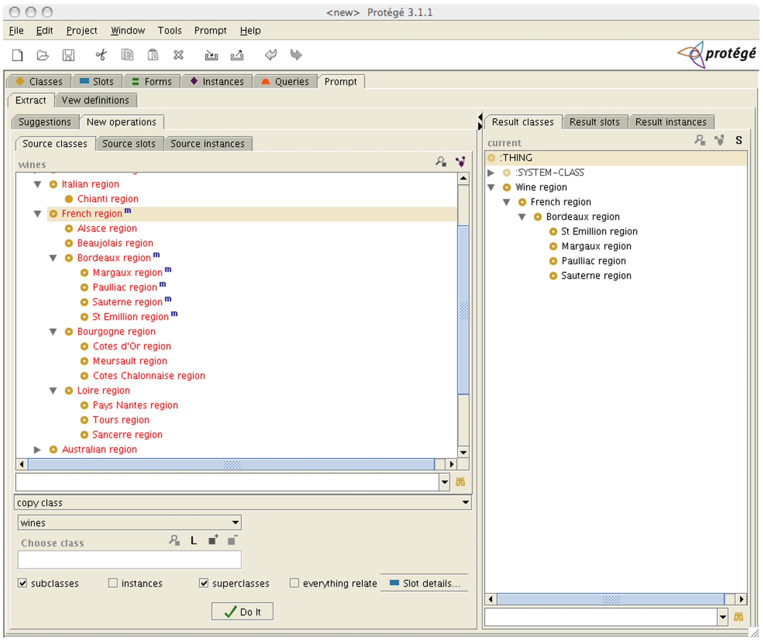


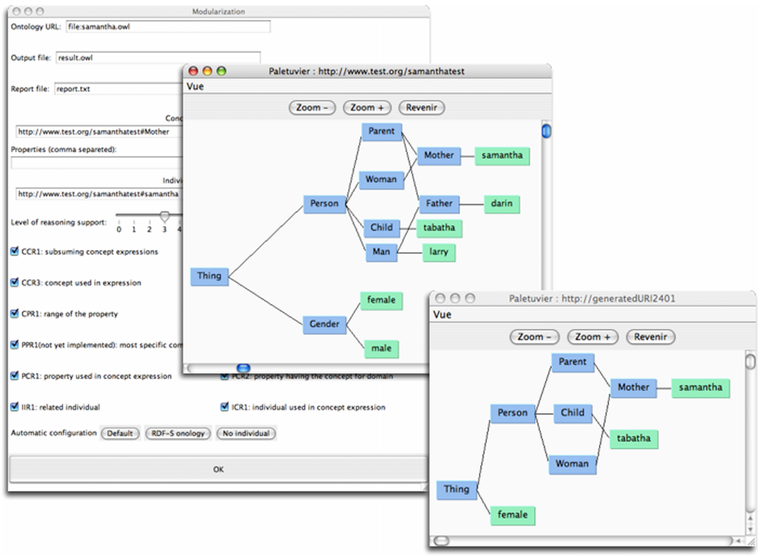
Fig. 3.4. Screenshot of the **Prompt** view extraction tool

Module Extraction Technique: Prompt

Prompt[6] is a toolkit, integrated as a plugin of the Protégé ontology editor[7] for comparing, merging and extracting views from ontologies. What we refer to as **Prompt** in the following corresponds to the part that concerns module extraction and that is called *traversal view extraction* in the Prompt toolkit [12]. This tool is designed as an interactive process for extracting sub-parts (modules, views) of ontologies to be integrated in the currently edited ontology (see Figure 3.4). A class of the source ontology is first selected by the user, using the standard navigation interface of Protégé. The principle of the approach is to recursively "traverse" the ontology relations from this class to reach other classes to be included. The relations to follow and the distances for which they have to be followed (the level of recursion) have to be first entered by the user. The whole process is incremental in the sense that new classes can be selected from the boundaries of the current module as new starting points for the extraction of other classes, which are added to the module, expending its boundaries.

Concerning the criterion on the *level of user interaction*, it is quite obvious that **Prompt** is not automatic at all: it requires the intervention of the user at each step of the process. The interactive nature of **Prompt** make it harder to evaluate (the resulting module is dependent on the user who defined it), but can also be seen as an

[6] `http://protege.stanford.edu/plugins/prompt/prompt.html`
[7] `http://protege.stanford.edu/`


Modularization

Ontology URL: file:samantha.owl

Output file: result.owl

Report file: report.txt

Con
http://www.test.org/samanthatest#Mother

Properties (comma separeted):

Indivi
http://www.test.org/samanthatest#samantha

Level of reasoning support: 0 1 2 3 4

☑ CCR1: subsuming concept expressions

☑ CCR3: concept used in expression

☑ CPR1: range of the property

☑ PPR1(not yet implemented): most specific co

☑ PCR1: property used in concept expression ☑ PCR2: property having the concept for domain

☑ IIR1: related individual ☑ ICR1: individual used in concept expression

Automatic configuration Default RDF-S onology No individual

OK

Fig. 3.5. Screenshot of the **KMi** module extraction tool

advantage, as **Prompt** is less dependent on a particular intuition on modularization, and so, less restricted in terms of usage scenarios. Concerning other application criteria, **Prompt** can be used on any ontology manageable by Protégé and generates modules (views) that are integrated in Protégé ontologies. The performance of the system in terms of time is obviously an irrelevant criterion for evaluating **Prompt**.

Module Extraction Technique: KMi

What is called here **KMi** is a technique developed at the Knowledge Media Institute (the Open University, UK) and described in [7]. It is also a "traversal approach", inspired from the two previous techniques for module extraction. One of the particularities of **KMi** is that it takes into account inferences during the modularization process, in order to validate some interesting logical properties. Moreover, **KMi** generally generates smaller modules (e.g. than **GALEN**) by taking shortcuts in the ontology hierarchy[8], while keeping all the necessary elements for describing the included entities.

KMi takes as an input only a sub-vocabulary of the source ontology, in the form of a set of class, property and individual names. Some other parameters can be modified through the interface (see Figure 3.5) but, since only the sub-vocabulary is required, **KMi** can be considered as fully automatic.

Moreover, **KMi** has been designed to work with – and exploit the content of – any kind of ontology, from simple taxonomies in RDF(S) to complex OWL structures.

[8] In particular, by including only the common super class of included classes, rather than the whole branches of super-classes.

Finally, as the resulting module is a standard, self-contained ontology, there is no need for particular tools to exploit it.

3.5.4 Results

Running the four modularization techniques on the three ontologies of the first example allowed us to test how they behave on simple, but yet practical real word examples. The second example concerns larger ontologies, with more heterogeneous levels of description. For example, **TAP** contains around 5500 classes, but no property or individual, whereas **Mid-Level** relies on almost 200 properties and is populated with more than 650 individuals for less than 2000 classes.

Evaluating the Modularizations

Analyzing the modules resulting from the considered modularization techniques is a way to better understand on which kinds of assumptions these techniques rely, and if these assumptions fit the requirements of the application.

Size

Figure 3.6 shows the *size* of the resulting modules for each system on Example 1 in terms of number of classes and properties (we did not include number of individuals as it is not a relevant indicator here). It can be easily remarked that **SWOOP** generally generates very large modules, containing 100% of the classes for two of the three ontologies, and an important proportion of the properties: in most of the cases, **SWOOP** generates one module with almost the same content as the original ontology. Because it has not been really configured for the experiment, **Prompt** also generates big modules. The tool developed in **KMi** is focused on generating modules

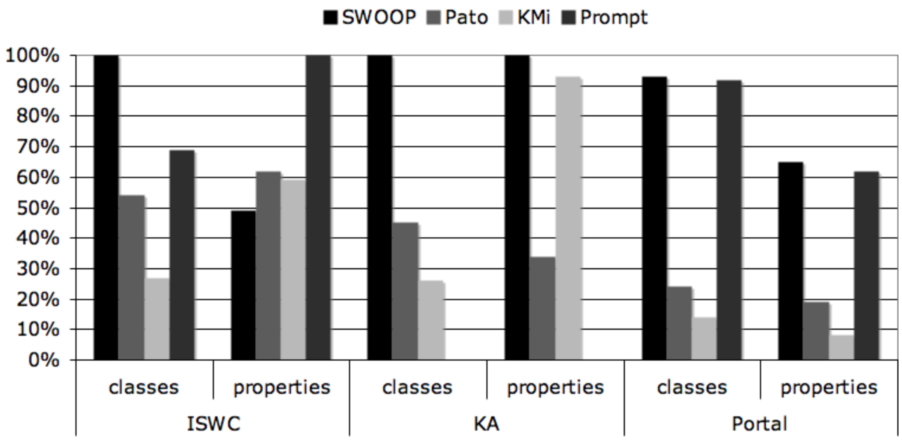


Fig. 3.6. Relative size of the resulting modules for the first example

with a small number of classes (the smallest), so that the ontology hierarchy would be easy to visualize. It nevertheless includes a large proportion of the properties, in order to keep the definition of the included classes intact. **Pato** is optimized to give an appropriate size. It generally operates an important reduction of the size of the ontology.

Concerning Example 2, the difference between **SWOOP** and other techniques is even more significant. Indeed, because of the poor structure of the considered ontologies (restricted uses of OWL constructs, few or insufficiently defined properties), **KMi** and **Pato** result in particularly small modules (less than 10 classes), whereas **SWOOP** still includes most of the content of the ontology in a single module.

Intra-module distance

The **KMi** tool relies on mechanisms that "takes shortcuts" in the class hierarchy for reducing the size of the module. Indeed, instead of including all the super-classes of the included classes, it only considers classes that relate these entities: their common super-classes. In that sense, the *distance* between the considered terms is also reduced in modules provided by **KMi**. For example, in the **Portal** ontology, by eliminating an intermediary class between *Researcher* and *Person*, **KMi** has reduced the distance between *Researcher* and *Student*, while keeping a well formed structure for the module. Since they do not include this kind of mechanisms, the other techniques generate modules in which the distance between included terms are the same as in the original ontology.

Logical criteria

SWOOP is the only tool that can guarantee the *local completeness* of the module. The focus **SWOOP** put on logical criteria can be an explanation of its unsatisfactory results concerning the size of the module. **KMi** has been designed to provide modules where a weaker notion of local completeness holds [7], which can be useful to facilitate the interpretation of the module by the user. This property generally holds also for other techniques.

Application Criteria

Application criteria takes an important part of the evaluation, as our goal is to integrate the modularization technique into an broader scenario having particular requirements. Table 3.1 summaries the evaluation of these criteria on the considered tools.

Level of interaction

As already mentioned, **SWOOP** is fully automatic and does not need any parameters besides the input ontology. As a module extraction tool, **KMi** requires, in addition to the source ontology, a set of terms from the signature of the ontology, defining the sub-vocabulary to be covered by the module. This sub-vocabulary corresponds to

Table 3.1. Evaluation of the application criteria: Assumption on the source ontology, level of required user interaction, availability of tools for manipulating modules, and performance on both Example 1 and Example 2

	Ontology	Interaction	Tool support	Perf. (Ex1/Ex2)
SWOOP	RDF(S)/OWL (in SWOOP)	Automatic	SWOOP/ Pellet	(few sec/sec-min)
PATO	RDF(S)	Parameters	OWL tools	(few sec/min)
Prompt	RDF(S)/OWL, Frame (in Protégé)	Interactive	Protégé	N/A
KMI	RDF(S)/OWL (parsed by Jena)	Automatic	OWL tools	(few sec/min)

the initial terms used for selecting the ontology: *Researcher*, *Student* and *University*. **Pato** has to be fine tuned with several parameters, depending on the ontology and on the requirements of the application. Here, it has been configured in such a way that modularizations in which the considered terms are in the same module are preferred. **Prompt** is an interactive mechanism, in which the user is involved in each step of the process. In particular, the class to be covered and the property to traverse have to be manually selected, requiring that the user has a good insight of the content of the ontology, can easily navigate in it, and that he understands the modularization mechanism. When using **Prompt**, we manually included the input terms and tried to obtain an (intuitively) good module, without going too deep in the configuration. Note that, since the system crashed at the early stage of the process, we did not manage to obtain results for the **KA** ontology with **Prompt**.

Assumption on the source ontology

In addition to what have been already mentioned concerning the type of ontologies the modularization tools are supposed to handle, it follows from the experiments that some of the techniques are not designed to take into account big and heterogeneous ontologies like the ones of Example 2. It is particularly hard for the user to handle the process of module extraction in **Prompt** when having to deal with several thousands of classes and hundreds of properties. We also did not manage to partition the **OntoSem** ontology using **Pato** because of the way **OntoSem** makes use of the `label` annotation property.

Moreover, the results obtained concerning the size of the modules in Example 2 shows that techniques are highly influenced by the inherent properties of the ontology to be modularized and that, in general, they *assume* a high level of description.

Performance

Apart from **Prompt** for which this criteria is irrelevant, each tool has only taken a few seconds or less on the *small ontologies* of Example 1. The ontologies in Example 2 are bigger and more heterogeneous. These elements obviously have an important impact on the performance of the modularization techniques: in the worst cases (**Pato** and **KMi** on **TAP**), it takes several minutes to get a modularization and none of the tested techniques can be used at run-time for such ontologies. However, as already observed in [14], loading and processing a big ontology generally takes longer than the actual modularization process.

3.6 Conclusion and Discussion

There is currently an important growth in interest concerning modularization techniques for ontologies, as more ontology designers and users become aware of the difficulty of reusing, exploiting and maintaining big, monolithic ontologies. The considered notion of modularity comes from software engineering, but, unfortunately, it is not yet as well understood and used in the context of ontology design as it is for software development. Different techniques implicitly rely on different assumptions about modularity in ontologies and these different *intuitions* require to be made explicit.

We have reported on preliminary steps towards the characterization of ontology modularization techniques. We reviewed existing modularization tools as well as criteria for evaluating different aspects of a modularization (logical, structural, application level), and used them on a particular scenario: the automatic selection of knowledge components for the annotation of Web pages. The main conclusion of these experiments is that the evaluation of a modularization (technique) is a difficult and subjective task that requires a formal, well described framework – a *benchmark* – taking into account the requirements of applications. Such a framework would be useful in two ways: first for application developers, it would provide a guide for choosing the appropriate modularization technique, and second, for the developers of modularization techniques, it would give directions in which techniques can be improved with respect to particular scenarios. More detailed conclusions are presented below, focusing on issues to be addressed for the evaluation of modularization techniques.

No Universal Modularization. As described in [7], the technique developed at **KMi** has been explicitly designed for the purpose of the knowledge selection scenario. Therefore, it is not really surprising that it obtained almost the best results for most of the evaluated criteria. Beyond the simple comparison of techniques, this result tends to demonstrate our original assumption: the evaluation of a modularization depends on the application requirements. Indeed, other scenarios may require more logical properties to hold, a better defined distribution of the module, or a more active involvement of the user, leading to the use of other techniques. It appears that techniques are designed to be used in particular scenarios, and so, that it is important to characterize them in terms of the requirements they fulfill to facilitate the development of applications relying on ontology modularization.

Existing Criteria are not Sufficient. In our experiments, we rely on criteria that have been explicitly used to evaluate different aspects of a modularization, with the underlying assumption that these aspects are relevant indicators of its efficiency, its usability or its relevance. However, when looking at the resulting modules, it seems obvious that important criteria are missing for evaluating the quality of a modularization. For example, in our scenario, it seems fundamental that the module keeps a *good structure* or that it maintains a *well defined* description of the included entities. It can be argued that it is the role of *logical criteria* to evaluate how formal properties are preserved in ontology modules, but it can be easily shown (at least experimentally) that they are insufficient to evaluate the *design quality* of a modularization. Integrating the evaluation of the quality (or rather qualities) of the produced modules is therefor an important task. We cannot expect modularization techniques to generate good quality modules from poorly designed ontologies, but, using these metrics, we can measure to which extent the inherent qualities of the source ontology are preserved in its modularization.

Techniques Need to be Improved. The difference in quality of the results for our second example (big, heterogeneous ontologies), compared to the first one (small, well defined ontologies), shows that, even if they are generally well designed and implemented, ontology modularization techniques need lots of improvements in terms of robustness, stability and scalability to be actually usable in real life scenarios. The evaluation of the considered criteria, taking into account different aspects of modularization, explicitly demonstrates that existing modularization techniques rely on different assumptions on ontologies and modularity of ontologies. Being restricted to particular use cases and ontologies prevent these techniques to be really usable in an environment like the Semantic Web, considering its inherent scale and heterogeneity.

References

1. Brewster, C., Alani, H., Dasmahapatra, S., Wilks, Y.: Data Driven Ontology Evaluation. In: Proceedings of International Conference on Language Resources and Evaluation (2004)
2. Chidamber, S.R., Kemerer, C.F.: A Metrics Suite for Object Oriented Design. IEEE Trans. Softw. Eng. 20(6), 476–493 (1994)
3. Cuenca Grau, B., Horrocks, I., Kazakov, Y., Sattler, U.: A Logical Framework for Modularity of Ontologies. In: Proc. of the International Joint Conference on Artificial Intelligence, IJCAI (2007)
4. Cuenca Grau, B., Parsia, B., Sirin, E.: Combining OWL ontologies using E-Connections. Web Semantics: Science, Services and Agents on the World Wide Web 4(1), 40–59 (2006)
5. Cuenca Grau, B., Parsia, B., Sirin, E., Kalyanpur, A.: Automatic Partitioning of OWL Ontologies Using E-Connections. In: Proc. of Description Logic Workshop (DL) (2005)
6. Cuenca Grau, B., Parsia, B., Sirin, E., Kalyanpur, A.: Modularity and Web Ontologies. In: Proc. of the International Conference on Principles of Knowledge Representation and Reasoning, KR (2006)
7. d'Aquin, M., Sabou, M., Motta, E.: Modularization: a Key for the Dynamic Selection of Relevant Knowledge Components. In: Proc. of the ISWC 2006 Workshop on Modular Ontologies (2006)
8. Dzbor, M., Domingue, J., Motta, E.: Magpie - towards a semantic web browser. In: Fensel, D., Sycara, K.P., Mylopoulos, J. (eds.) ISWC 2003. LNCS, vol. 2870, pp. 690–705. Springer, Heidelberg (2003)

9. Loebe, F.: Requirements for Logical Modules. In: Proc. of the ISWC 2006 Workshop on Modular Ontologies (2006)
10. Lopez, V., Sabou, M., Motta, E.: Ontology Selection on the Real Semantic Web: How to Cover the Queens Birthday Dinner? In: Staab, S., Svátek, V. (eds.) EKAW 2006. LNCS, vol. 4248, pp. 96–111. Springer, Heidelberg (2006)
11. MacCartney, B., McIlraith, S., Amir, E., Uribe, T.E.: Practical Partition-Based Theorem Proving for Large Knowledge Bases. In: Proc. of the International Joint Conference on Artificial Intelligence (IJCAI) (2003)
12. Noy, N.F., Musen, M.A.: Specifying Ontology Views by Traversal. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 713–725. Springer, Heidelberg (2004)
13. Schlicht, A., Stuckenschmidt, H.: Towards Structural Criteria for Ontology Modularization. In: Proc. of the ISWC 2006 Workshop on Modular Ontologies (2006)
14. Seidenberg, J., Rector, A.: Web Ontology Segmentation: Analysis, Classification and Use. In: Proc. of the World Wide Web Conference (WWW) (2006)
15. Stuckenschmidt, H., Klein, M.: Structure-Based Partitioning of of Large Concept Hierarchies. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 289–303. Springer, Heidelberg (2004)
16. Tartir, S., Budak Arpinar, I., Moore, M., Sheth, A.P., Aleman-Meza, B.: OntoQA: Metric-Based Ontology Quality Analysis. In: IEEE ICDM 2005 Workshop on Knowledge Acquisition from Distributed, Autonomous, Semantically Heterogeneous Data and Knowledge Sources (2005)
17. Yao, H., Orme, A.M., Etzkorn, L.: Cohesion metrics for ontology design and application. Journal of Computer Science 1, 107–113 (2005)

4

On Importing Knowledge from Ontologies[*]

Alexander Borgida

Dept. of Computer Science,
Rutgers University, USA
borgida@cs.rutgers.edu

Summary. This chapter focuses on the notion of importing terms from an ontology. Rather than proposing a specific formalism for this task and proving theorems about its properties, it starts by surveying a sample of lesser known papers on this topic in both the AI and Database literature. Based on this, it then derives a list of desirable properties for the notion of *"importing a set of identifiers from an exporting to an importing knowledge base"*, and provides a framework for alternate formal definitions of this notion. Some of the more novel aspects are the idea of modifying the exporting ontology before the subset of axioms to be imported is determined, and limiting the use of imported terms. The chapter concludes with a review of the concept of "expressive power", and how it might apply to importing mechanisms.

4.1 Introduction

As indicated in Chapters 1 and 3, there are multiple reasons why one might be interested in modularity aspects of an ontology. One reason is to provide units of independent development: the ability to write and especially modify a large ontology in smaller, easier to understand chunks. In this chapter, we are concerned with a second aspect: the desire of the developer of some ontology KB$_{impt}$ to reuse information about the meaning of some set of identifiers Ω={ $N_1, N_2, \ldots$} which has already been captured in ontology KB$_{expt}$ — a task we shall call *importing knowledge*.

One can begin by trying to identify some general approaches to this problem, supported in part by an analogy with similar notions in programming languages, where identifiers of types, methods, and constants (corresponding to concept, property and individual identifiers) might need to be imported from other implemented programs (ontologies).

1. Include the whole KB$_{expt}$ in KB$_{impt}$. This seems to be the "gold standard" as far as inferences that one would want to be able to draw from the enhanced

[*] Some material in this chapter has been presented at the WOMO'06, DL'07 and WOMO'07 workshops. This research was supported in part by the U.S. DHS under ONR grant N00014-7-1-0150.

KB_{impt}, but has as significant downside the explosive increase in the size (and corresponding decrease in comprehensibility) of the importing ontology. It corresponds loosely to the `#include` preprocessor command in the C language. The main obstacles to be overcome by such a mechanism are name conflicts (what to do if the same name appears in multiple files), and circularity (avoiding infinite loops in case files directly or indirectly include each other).

 We note that in programming languages such as C, header files play an additional important role for independent compilation, and provide an abstraction of the implementation of functions into their type signatures — ideas which as of yet have not been exploited in ontology modules.
2. Partition/segment ontologies into independent modules, and then have KB_{impt} import only those modules that involve names in Ω. This corresponds to the case of programming languages like Modula and Ada, where one imports entire modules, rather than particular procedure identifiers for example. Note that in these languages each module has exactly one implementation. (Once again, programming languages have additional important notions, namely information hiding.)
3. Rather than relying on pre-existing modules, investigate the theory of an operator import(Ω,KB_{expt}) or even import(Ω,KB_{expt},KB_{impt}), that takes into account the set of identifiers Ω and (properties of) the ontologies involved, to tailor-fit the material to be imported. For a parallel in programming languages, one might think of a *Java interface* as corresponding to the signature Ω of interest. An interface can be implemented by alternative classes, which would correspond to alternate exporters, which may provide additional public methods. The user of the interface (the importer) then chooses not to use any of the additional public identifiers made available by the exporting class, limiting himself to Ω. This distinction between the exporter and the needs of multiple importers is even clearer in Python, where one can write a statement such as `from YourModule import name1 as name2, name3 as name4,...` to use just some of the public identifiers defined in `YourModule`, and even rename them locally.

The aim of this chapter is to determine some of the desirable properties of the import operator mentioned in item 3 above, and to look at some alternative formal definitions for it.

4.2 Examples of Knowledge Importing

We begin by reviewing a number of proposals for knowledge import in the Artificial Intelligence and Database literature, with the goal of extracting interesting intuitions about importing. The original papers offer varying degrees of detail and formality. We opt for using pseudo-code for the ones which are more informal, with the aim of making clear the source of the intuitions we summarize in the last subsection.

4.2.1 Importing Non-logical Symbols in Ontolingua

Ontolingua[21] was developed as a knowledge representation language, as part of the first serious effort at studying large-scaleontology development, and is based on First Order Logic, though it has some aspects which appear to be higher order. To be specific, we will consider a specific ontology expressed in Ontolingua: the medical ontology ON9.3 [1] developed at CNR in Italy as part of a decade long project on integrating medical ontologies [15]. It is one of the largest detailed ontologies available publicly, having complex axioms expressed in KIF – a variant of First Order Logic augmented with notations from frame representations (classes, slots). This ontology is organized in close to 50 modules (called *"theories"*), almost all of which import terms from other modules. For example, the `Anatomy` theory, which specifies 55 concepts, lists in its documentation

```
Theories included by Anatomy:
   Meronymy,
   Positions,
   Topo-Morphology
```

which are cases of ordinary module inclusion. In addition, the description of `Anatomy` also contains statements of the form

```
The following constants were used from included theories:
   * 3d-Area-Of defined as a relation in theory Topo-Morphology
     ... (60+ other terms)
```

These (redundant) declarations explicitly document the specific identifiers used from the modules imported. In this case, the predicate `3d-Area-of` can be used as part of the axioms expressed in `Anatomy`. Ontolingua assumes that each module defines non-logical constants (predicates) which have globally unique names, thus avoiding problems due to name clashes and circularity of inclusion, but allows these constants to be given more convenient surface names. There are then rules for visibility of modules and of identifiers within modules.

Using the taxonomy of approaches for importing mentioned at the end of Section 4.1, these are examples of approach (2): e.g., the axioms in `Topo-Morphology` were imported into `Anatomy`.

More interestingly, the following statement appears in the documentation of the `Anatomy` theory:

```
The following constants were used from theories not included:
  * Anatomical-Abnormality defined as class in theory Abnormalities
  * Antigen defined as a class in theory Biologic-Substances
  * Bacterium defined as a class in theory Natural-Kinds
  * Binds-With defined as a relation in theory Molecular-Biology
    ... (20+ other terms)
```

This suggests that certain constants were imported into the theory without explicitly including their entire modules — approach (3) above. If the modules from which these 27 terms come would also have been included, the total number of theories included would have risen from 3 to 14, so it seems the authors felt an intuitive need to not import full modules when only a few constants from them were needed. This is not an isolated occurrence, and is present even in the original KIF ontology libraries, such as `KIF_Lists`[1]. Unfortunately, no special semantics for this construct is offered; instead, reference [21] states

> That is, if an ontology `A` contains an axiom that references a symbol in the vocabulary of an ontology `B`, then the system implicitly considers `B` to be included in `A`. This inclusion rule ensures that the axioms specifying the "meaning" (i.e., restricting the possible interpretations) of the referenced symbol are a part of the ontology in which the reference occurs. A more refined rule could include only those axioms that could affect the possible interpretations of the symbol, but we have not developed such a rule.

Therefore, in Ontolingua, approach (3) reduces to approach (2).

Despite the current absence of a semantic distinction in Ontolingua, we take the introduction of a syntactic differentiation between importing identifiers and using names from imported modules as evidence of the pragmatic usefulness of the notion, especially for large, complex ontologies.

4.2.2 Importing Taxonomy from WordNet

Navigli [27] is interested in situations where several small terminologies with *local, domain-specific terms* have been extracted (possibly semi-automatically, using text processing techniques). These need to be integrated, and one way to do so is by tying them together through a general ontology, such as WordNet, where terms are already organized in a IsA (hyperonym/hyponym) hierarchy. In particular, assuming that the local concepts are organized in their own taxonomies already rooted at terms t_i, the first step (not discussed here) is to find closest unambiguous meanings (synsets) m_i in WordNet, and adding IsA links from t_i to m_i. The remaining problem is that there are very many terms in the resulting joint terminology which are irrelevant to the task at hand. Instead, what is wanted is a small subset of terms in WordNet that subsume and tie together the terms in Ω=$\{m_1, m_2, \ldots\}$ — a subset we shall write as import(Ω,WordNet).

The algorithm proposed in [27] starts from the nodes for Ω, and constructs an initial, pruned set of terms:

- S_1 := IsA*(Ω); */* Nodes in WordNet that subsume Ω.*[2] */

In earlier work, Swartout et al [32] used just this set for import(Ω,WordNet), projecting IsA$^+$ over it.

Navigli, on the other hand, believes that this set may contain many concepts of dubious utility, and proposes to further prune the result, by defining the following additional sets:

[1] `http://www-ksl.stanford.edu/knowledge-sharing/ontologies/html/index.html`

[2] We use IsA$^+$ to refer to the transitive closure of the IsA relation, and IsA* to the reflexive transitive closure. For binary relation ρ, we use $\rho(x)$ to refer to the set $\{y \mid \rho(x, y)\}$ when x belongs to the domain of ρ, and generalize this to sets: $\rho(T) = \bigcup_{x \in T} \rho(x)$.

- S_2 := {nodes in S_1 that have 2 or more children (hyponyms) in S_1}
- S_3 := {children of nodes in S_2 that belong to S_1}
- S_{top} := {nodes in S_1 that are at the very top of WordNet (top two levels)}

The final imported fragment of WordNet then consists of the concepts $\Omega \cup S_{top} \cup S_2 \cup S_3$ together with all IsA edges between these *entailed* by the original WordNet taxonomy.

The motivation for including S_2 (actually, for excluding the complement of S_2) is that *"a node with only 1 hyponym gives no additional information and provides no additional classification"*, while the reason for including elements in S_3 is *"avoid collapsing the hierarchy"*.

If one thinks of WordNet as a set of axioms $\mathcal{T}$ in some logic such as FOPC, consisting of atoms of the form $IsA(b, c)$, then one aspect of this scheme worth noting is that the material imported is not strictly a subset of the set $\mathcal{T}$, but rather of its logical consequences: $IsA(d, e)$ is imported when term d is specialization of e, but the intermediate concepts are eliminated (because they only have one child in set S_1).

4.2.3 Knowledge Bus

Knowledge Bus [28] is a system for generating an Information System, including its database schema and API, from a focused subset of the Cyc ontology [22]. The advantage of this technique is that it facilitates later integration of databases derived from the same ontology, in contrast to the complex merger of independently developed Information Systems, which requires costly and difficult reverse engineering. More precisely, the ultimate goal of this project is to take an initial seed set of terms, and generate from it a deductive database implemented in the XSB system, and encapsulated in Java. Note that Cyc contains much more than just concept hierarchies: there are general predicates, and formulas in CycL (a variant of FOL), as well as default reasoning rules.

The idea of the approach in [28] is to first identify all axioms which might contribute to proofs about instances of concepts in the seed set of identifiers S_0 (which correspond to our set Ω). The algorithm relies on two notions:

- each predicate p in Cyc has a signature $signature(p) = T_1 \times \ldots \times T_k$, which indicates that the j-th argument of p must belong to concept T_j;
- the assumption that multiple occurrences of a variable (which in some sense perform "joins", as in the case of y in the axiom $p(x, y), q(y, z) \rightarrow r(x, z)$) requires the corresponding arguments of the predicates to be type-compatible (actually, in this case IsA related).

Therefore, starting from the set S_0, the algorithm computes by a fix-point iteration (using operators Π and Γ below) a final set of concepts of interest S_*, and associated predicate names P_*. The set of axioms retrieved are those involving only predicates in P_*.

The operators Π and Γ are defined as

- for a set of concepts S, $\Pi(S) = \{p \mid signature(p) = T_1\times\ldots\times T_k \wedge (\exists j)\ [T_j \in S \wedge \neg dataType(T_j)]\}$ */* Gather predicates that have some argument typed by a concept in S. These predicates can be "joined" to each other in a formula by using a variable x whose type is in S. */*
- for a set of predicates P , $\Gamma(P) = \{T \mid (\exists p)\ p \in P, signature(p) = T_1 \times \ldots \times T_k \ \wedge\ (\exists j)\ IsA^*(T_j, T)\}$ */* Gather generalizations of concepts used to type arguments of predicates in P. */*

A second problem that needs to be addressed at this point is translating the selected Cyc axioms into the language of the deductive database. This involves translating general non-Horn clauses into directional Horn rules and integrity constraints, dealing with time, defaults. etc. Although we do not consider the details here, this points out both an interesting and significant general issue in knowledge importing: the need to consider the possibility that the logic of KB$_{impt}$ may not be compatible with that of KB$_{expt}$.

4.2.4 Pruning Ontologies for Database Conceptual Design

Conesa and Olive [14] start from a list of terms (entity and relationship names) that are of interest in a particular Information System application domain. This list may be developed by analyzing the software requirements — names appearing in procedure parameters, pre- and post-conditions. (This is the set Ω in our case.) They wish to enrich it with integrity constraints derived from some large existing general ontology, in this case OpenCyc translated into UML — the material to be imported.

In [13], the task is *specified* as finding a *minimal* sub-ontology O_P of the exporting ontology O_X that contains Ω, and all formulas/constraints in the set

Axioms1 := {formulas ψ in O_X such that $vocab(\psi) \subseteq IsA^*(\Omega)$ }
while also preserving all IsA$^+$ relationships derivable among these in O_X. Although not explicitly stated, we presume that the formulas in Axioms1 are of interest since, by inheritance, they also "talk about" (quantify over) instances of terms in Ω. Note that the set of axioms selected here is much smaller than in Section 4.2.3 since *every* symbol in ψ must be in $IsA^*(\Omega)$, rather than allowing an arbitrary potential chain of resolutions/inferences involving these classes. This reflects the use of Axioms1 for integrity checking rather than deduction.

The actual algorithm presented in [14, 13] is considerably more complex, and can again be abstracted as a sequence of set specifications:

- $S_{SurelyNeeded}$:= Ω ∪ vocab(Axioms1). */* This subset of terms from IsA*(Ω) are ones definitely wanted. */*
- S2 := $\{\, y \mid (\exists x, z \in S_{SurelyNeeded}).IsA^*(x, y) \wedge IsA^*(y, z)\}$ */* These are all the terms in $S_{SurelyNeeded}$ or on inheritance paths between them. They appear to be needed in maintaining the IsA relationship between elements of $S_{SurelyNeeded}$. */*
- S3 := remove from S2 concepts not in $S_{SurelyNeeded}$, as long as (i) the existence of *some* subclass path between pairs of concepts in $S_{SurelyNeeded}$ is preserved,

(ii) the concepts removed have only single parents and children. /* *The concepts removed only provide redundant inheritance paths. Note that condition (ii) is suggestive of Navigli's trimming condition.* */

It may not be clear why the algorithm doesn't simply eliminate all terms not in $S_{SurelyNeeded}$, and project the IsA relationship on it, as in Navigli's proposal. It turns out that in addition to the pruning phase described above, Conesa and Olive also suggest a refactoring phase, where, among others, concepts with single children or parents are collapsed with their neighbors.

4.2.5 MOVE

One of the aims of the MOVE project [33] is to derive a sub-ontology starting from a UML conceptual model, given two sets: Ω — identifiers that are definitely wanted in the sub-ontology; and $\Omega_{excluded}$ — identifiers definitely not to be included; the remaining identifiers may or may not be included in the final ontology view depending on which policies are adopted.

One of the interesting aspects is that MOVE permits some attribute name, e.g., `phone#`, to be selected as part of Ω, without the class on which it originally appears, e.g., `OFFICE`, also being selected. In such cases, MOVE seems to offer at least three choices:

1. the concept `OFFICE` is also added to the sub-ontology;
2. `phone#` appears on a subclass of `OFFICE` that is included in the sub-ontology;
3. `phone#` is included *indirectly* as part of a new composite relationship. For example, suppose that class `EMPLOYEE` is selected in Ω, and it is related by association `worksIn` to `OFFICE`. Then one could create a new relationship `worksIn_Office_having_phone#`, representing the composition of `worksIn` with the link `OFFICE --- phone#`, and make this part of the derived sub-ontology. This option is conditional on the requirement that the resulting association be guaranteed to be functional (e.g., the maximum cardinality of `worksIn` and `phone#` are 1).

The choice between the above options, and others like them, is made on the basis of so-called "optimization schemes", which represent different kinds of policies for removing non-selected nodes and edges in the diagram. For example, the Total Simplicity Optimization Scheme (TSOS) will result in the smallest possible solution that is still a valid ontology. (The validity rules include ones requiring an ontology to be connected, for example.) Another policy, "semantic completeness", requires among others that if a class is selected in Ω, then all of its super-classes will also appear in the final result.

Option 3 above is a noteworthy novelty of this proposal, because it relies on the ability to *expand the definitions available* in the ontology. To support this, the algorithm investigates all *paths* in the UML graph connecting concepts in Ω, and even has schemes for suggesting appropriate names for such new composite relationships.

4.2.6 Importing/Modularizing Based on Role Identifiers

A proposal by Herzig and Varzinczag [24] seems to be based on the intuition that roles are the more stable part of an ontology, in contrast to concepts, and therefore may be a more appropriate basis on which to modularize it or make decisions for importing.[3]

While this is a proposal for partitioning ontologies into modules, thus belonging to Part II of this book, we survey it here because it exhibits an interesting feature in common with many of the above proposals: the original exporting knowledge base is modified before the modules are determined.

In general, after introducing the notation

KB^R = { axioms in KB that involve roles in set R }
KB$^\emptyset$= { axioms in KB that do not involve any roles }

the authors consider whether an ontology has the desirable property (called *"modularity property"*) that for each φ

$$KB \models \varphi \text{ iff } KB^\emptyset \cup KB^{roleNames(\varphi)} \models \varphi$$

i.e., the only axioms necessary to reason about a formula are those involving the roles appearing in them, as well as those involving only concepts. This provides a basis for deciding what axiom from KB to include in the material imported for Ω: those involving roles in Ω, plus those in KB$^\emptyset$.

Of course, most knowledge bases do not have the "modularity property". Herzig and Varzinczag [24] focus on Description Logic knowledge bases expressible as $\mathcal{ALC}$ TBoxes $\mathcal{T}$, and investigate how such a $\mathcal{T}$ can be tested to see whether it has this property, and if not, how to modify it so the property holds.

Interestingly, the general question of having the modularity property reduces to a question concerning only so-called boolean subsumptions (ones not involving roles): are there such subsumptions derivable using axioms involving roles which are not derivable only from the explicitly listed boolean axioms of the ontology, $\mathcal{T}^\emptyset$? Formally, it is shown that if there is no boolean subsumption judgement φ such that

$$\mathcal{T} \models \varphi \text{ but } \mathcal{T}^\emptyset \not\models \varphi$$

then the TBox $\mathcal{T}$ has the modularity property in the above sense. Not only does the paper provide an algorithm for testing for this property, but if the property does not hold, then it provides a set of boolean subsumptions $\mathcal{T}^\emptyset_{implicit}$, such that when these are added, the entire knowledge base $\mathcal{T} \cup \mathcal{T}^\emptyset_{implicit}$ is indeed modular. The paper argues that the presence of the implicit axioms in $\mathcal{T}^\emptyset_{implicit}$ is in fact a sign of poor knowledge engineering, and makes a relatively convincing case that making them explicit (or correcting $\mathcal{T}$ so the formula is not derivable) is a reasonable approach to modifying $\mathcal{T}$ so that it is modular.

[3] This intuition is stronger in epistemic modal logics, which are formally equivalent to the $\mathcal{ALC}$ DL, where the beliefs of each believer can be reasonably separated.

Unfortunately, the above nice theory only works for TBoxes which have a special restricted syntax: every axiom can involve at most one role identifier, so that $\{\mathcal{T}^{\emptyset}\} \cup \{\mathcal{T}^{\{r_i\}} \ : \ r_i$is some role identifier$\}$ actually *partitions* $\mathcal{T}$. At the very least, this requires taking the most common kinds of ontology axioms, having the form $A \sqsubseteq \exists p.B \sqcap \exists q.C \sqcap \ldots$, and replacing them by the set $\{A \sqsubseteq \exists p.B\ ,\ A \sqsubseteq \exists q.C\ ,\ \ldots\}$ thereby expanding conjunction. To deal with more general cases, including nested restrictions, it is possible to mechanically put a set of axioms into such a form by introducing a new concept identifier F for every role restriction $\forall r.C$ (resp. $\exists r.C$) encountered, replacing the restriction by F in existing axioms, and adding a definition equating F and the restriction: F $\equiv \forall r.C$ (resp. F $\equiv \exists r.C$).

Unfortunately, it is likely that such modifications of $\mathcal{T}$ produce a result that is not understandable to knowledge engineers since names like F have no intuitive meaning.

4.2.7 Conclusions of the Survey: Some Intuitions Concerning Importing

We summarize here some of the intuitions apparent in the papers reviewed in the preceding subsections.

Content of what is to be imported

In general, it seems that when asked to import the meaning of identifiers in Ω from KB$_{expt}$, one is expected to behave as if all of KB$_{expt}$ was included in KB$_{impt}$. The material to be imported should at least be "sound" (i.e., it should include only axioms about Ω that are logical consequences of KB$_{impt}$). But other desiderata below may over-ride the need/possibility for "completeness"; i.e., not every formula involving only Ω that is provable in KB$_{expt}$ must be provable from the material imported. Nor are there uniform requirements that when a concept is imported, so should be all of its superconcepts—a feature of several modularization schemes.

Cutting down on what is imported

There are several reasons why it is undesirable to import all of KB$_{expt}$:

- The result may be too large *for the purpose of understandability* by the humans developing KB$_{impt}$; this motivation is most evident when starting from very large general ontologies, such as Cyc or WordNet, or when importing only one or two terms from a module in an Ontolingua ontology (see Section 4.2.1).
- The result may be too large *for the purposes of machine reasoning*; this problem arises when the imported knowledge is cached locally, as in [31], since in most cases the complexity of reasoning rises as an (often exponential) function of the size of the local knowledge base. Again, the size and complexity of reasoning with Cyc in [28] make this point evident.

Therefore we enunciate

MinPrinciple: *Minimize the amount of material actually imported.*

Expanded vocabulary $\Omega_\oplus$

In order to understand the names in Ω, its terms need to be inter-related, and this may require an enlarged set of symbols, which we shall call $\Omega_\oplus$. For example, to understand `Dog` and `Cat`, it is reasonable to assume that their closest superclass from KB$_{expt}$, `Animal`, is also added. (This feature appears in many of the proposals reviewed above.) Less obviously, $\Omega_\oplus$ may contain not just atomic identifiers from KB$_{expt}$, but also *derived concepts/relationships* built from them, as illustrated in Section 4.2.5, where `Person` is related to `Phone#` by `officeOf.hasPhone`. By assigning an identifier to such a composite term (as in [33, 31]), one gets a different behavior than importing the individual identifiers used in the definition, since these can no longer be combined in arbitrary ways in KB$_{impt}$.

In line with *MinPrinciple* above, many of the proposals we have seen try to minimize the size of $\Omega_\oplus$. For example, Navigli [27] eliminates most concepts with only one child of interest, while Conesa and Olive [14] remove many upper-level concepts that do not contribute axioms that talk *only* about IsA$^*(\Omega)$. This makes sense in the context of an upper-level ontology such as DOLCE [23]: if some specific domain ontology only needs the concepts `AgentivePhysicalObject` and `NonAgentivePhysical-Object`, then it seems rather an over-kill to force the user to understand, in addition to `PhysicalObject` (which is the immediate parent of these two), additional super-concepts such as `PhysicalSubstantial`, `Substantial`, and `Endurant`, which subsume `PhysicalObject`.

We remark that, as shown in [8], the effort of minimizing the number of additional concepts introduced, may in fact be a quite considerable part of the cost of computing the right axioms to import, because the problem is NP-complete.

Also, the technique of importing a single re-named composite relationship, described in Section 4.2.5, provides an additional approach to the minimization of $\Omega_\oplus$. In fact, it raises the hitherto unexplored possibility that the terminology of KB$_{expt}$ be expanded with new definitions for the purpose of minimizing the material to be imported — a problem related to the minimal rewriting of concepts using definitions [3].

4.3 Source of Axioms to Be Imported

Before going on to consider the spectrum of approaches to formalizing the *content* returned by import, we take a closer look at the issue of the *source* of the actual axioms imported. There are at least three possible approaches, examined in subsections below.

4.3.1 The Syntactic-Subset Approach

One obvious approach, taken by many recent proposals for modularization and import, is to import some (minimal) *subset of the axioms* in KB$_{expt}$. This has certain conceptual advantages: simplicity, and, in some cases, clear evidence for the existence of a minimal set of axioms to import: Start with KB$_{expt}$ and repeatedly remove axioms until no more can be removed without losing the desired property of describing the meaning of terms in Ω (no matter how this is specified). However, there are also several difficulties with this approach:

- It is subject to the vagaries of the syntactic specification by knowledge engineers: if $\Omega=\{A, B\}$, then one imports different things from the following logically equivalent DL knowledge bases: $\mathcal{T}_1=\{A\sqsubseteq (B\sqcap C)\}$, $\mathcal{T}_2=\{A\sqsubseteq B, A\sqsubseteq C\}$.
- As a corollary of the above example, one may end up importing more than needed (e.g., from $\mathcal{T}_1$), thus contradicting the spirit of *MinPrinciple*.

4.3.2 The Logical Consequence Approach

At the other extreme, one may be allowed to import all the formulae that are logical consequences of KB$_{expt}$, and which involve only symbols from Ω. The disadvantage of this approach is that such a set may not be found for some logics, and even when it exists, characterizing such an infinite set in a finite way may be quite cumbersome (e.g., exponentially many logical implicants). In addition, both the meaning and motivation for such automatically derived formulae may be totally obscure to knowledge engineers. Using the terminology of [17], one loses *"intelligibility"*: the need for ontology engineers to be able to make sense of material imported.

4.3.3 The Expanded Axioms Approach

By choice, none of the proposals we have reviewed in Section 4.2 fit either of the above cases. We believe that they can be explained in part by an intermediate position which allows a limited set of modifications to the exporting knowledge base, before the set of axioms to be imported is determined by the syntactic subset approach. This technique can be seen explicitly in Section 4.2.5, where new definitions desired are added to the exporting knowledge base, and in Section 4.2.6, where the axioms are split up so they involve only one property, e.g., replacing axiom $Person\sqsubseteq (\forall eats.Meat\sqcap \forall drinks.Wine)$ by $\{\ Person\sqsubseteq \forall eats.Meat,\ Person\sqsubseteq \forall drinks.Wine\ \}$. And one can see it being present implicitly in Sections 4.2.2 and 4.2.4, where, for example, if the ontology contains $\{IsA(Dog, Canine), IsA(Canine, Animal)\}$, one may import only $\{IsA(Dog, Animal)\}$ when $\Omega=\{Dog, Animal\}$, which can be viewed as expanding KB$_{expt}$ so it contains all deducible concept name subsumptions (i.e., IsA$^+$, rather than just IsA).

Interestingly, several of these examples are related to research on the problem of *explanation* in Description Logics [26]. In particular, the following inference steps have been found to be *un-necessary* to present to users explicitly, since they tend to find them obvious:

- from $E1\sqsubseteq E2$ and $E2\sqsubseteq E3$ deduce $E1\sqsubseteq E3$
- from $E1\sqsubseteq (E2\sqcap E3)$ deduce that $E1\sqsubseteq E2$
- from $A\equiv E$, where A is an identifier, deduce that $A\sqsubseteq E$

Formally, explanations are normally thought of as minimal-length proofs from TBox $\mathcal{T}$, using some proof-theory whose rules of inference are easily comprehensible by users. (See [10] for more details on the choice of such proof theories.) The elimination of un-necessary proof steps, leading to streamlined explanations, can be modeled by augmenting $\mathcal{T}$ with "trivial lemmas" — logical consequences that are obvious and would need no explanation. (Let us use obvious($\mathcal{T}$) to denote this augmented TBox.) Note that the notion of what constitues an "obvious consequence" is empirical, and depends not only on the particular logic $\mathcal{L}$ and its proof theory, but possibly even on the application.

In [8], it was therefore suggested that the set of axioms to be imported should form a subset of obvious(KB_{expt}), rather than KB_{expt}. Note that streamlined explanations in ($\mathrm{KB}_{impt} \cup \mathrm{KB}_{expt}$) will be preserved in ($\mathrm{KB}_{impt} \cup$ import(Ω,KB_{expt})). The operator obvious, which should then become an explicit parameter of the import() operator, is a particularly good candidate for expanding KB_{expt} because it preserves "intelligibility" [17].

4.4 Approaches to Formalizing Import

There are a surprising number of different ways in which one can formalize the notion of importing. The following is an attempt at systematically covering the choices in the reviewed papers, and at evaluating them in light of the intuitions in Section 4.2.7. It may be worth pointing out that with the exception of [24], none of the papers reviewed have formal proofs concerning the soundness or completeness of the material imported. In some cases this is relatively easy to determine by inspection, but in other cases, involving complex logics like CycL, it would be a much harder issue.

4.4.1 Importing Modules

A number of previous proposals, many of which are described in Part II of this book, have suggested partitioning an ontology into modules $\{M_i\}$, dealing with different concepts and relationships. In this case, presumably the natural specification of the set of axioms KB_{sub} to import if one desires to learn about Ω would be the union of axioms appearing in modules whose vocabulary overlaps with Ω, i.e.,

$$\mathrm{KB}_{sub} = \bigcup_{vocab(M_i) \cap \Omega \neq \emptyset} M_i$$

This is a syntactic subset approach according to the taxonomy in Section 4.3, with the concomitant advantages (understandability) and disadvantages. It supports expansion of the vocabulary, because other names appearing in each module are also added, so that $\Omega_{\oplus}$ is the union of the vocabularies of all modules containing some element of Ω. Probably the biggest problem with this approach is the fact that it does not take seriously enough *MinPrinciple*. For example, many modularization proposals have the property that if concept name C appears in a module, so do *all* its super-concepts; yet we have seen that several of the examples of importing reviewed earlier, in Section 4.2, explicitly eschew this property.

4.4.2 Importing as an Extraction Operator

The most obvious logical approach is to view import as a binary operator import$_2(\Omega$, KB$_{expt})$, focusing on what can be deduced about Ω in KB$_{expt}$, namely the set

$$E^{\Omega} = \{\varphi \mid vocab(\varphi) \subseteq \Omega \text{ and } \mathrm{KB}_{expt} \models \varphi\}$$

The options considered in Section 4.3 apply here to yield the following:

- syntactic subset approach: Find a sufficiently large subset KB$_{sub}$ of KB$_{expt}$, such that it entails everything in E^{Ω}, though it will likely entail other formulas too.
- expanded axioms approach: Expand KB$_{expt}$ with some clearly specified set of additional axioms (e.g., obvious(KB$_{expt}$)), to yield ${\mathrm{KB}_{expt}}^{expanded}$, and then apply the preceding.
- logical consequence approach: Find some valid (but finite) knowledge base KB$_{sub}$, with vocabulary Ω, whose contents are entailed by KB$_{expt}$ and whose logical consequences are *precisely* E^{Ω}.

Let us now consider the other desiderata we have identified above.

In order to apply *MinPrinciple*, the set KB$_{sub}$ should be minimal in the sense that the removal of any axioms would lead to the loss of the defining property of KB$_{sub}$. Unfortunately, there is no formal guarantee that there is only one minimal set. This is most obvious in cases when there is some redundancy in KB$_{expt}$, or there are multiple ways of deducing some fact. For example, if KB$_{expt}$ contains $\{\ Cat \sqsubseteq Feline,\ Feline \sqsubseteq Animal,\ Cat \sqsubseteq Pet,\ Pet \sqsubseteq Animal\}$, and we have $\Omega = \{Cat, Animal\}$, then in a syntactic subset approach, $\{Cat \sqsubseteq Feline$, $Feline \sqsubseteq Animal\}$ and $\{Cat \sqsubseteq Pet,\ Pet \sqsubseteq Animal\}$ are two alternative minimal candidates for KB$_{sub}$. Unfortunately, in such cases one is left with three unpleasant choices: (a) proceed totally arbitrarily, choosing one of the sets; (b) take their union, abandoning minimality; (c) take their intersection, thereby possibly missing important parts of E^{Ω}. Proposals for importing based on *a priori* segmentation into modules often have the effect of taking approach (b), since they group together the axioms that lead to the same conclusion.

We also note that one could apply *MinPrinciple* to the choice of $\Omega_{\oplus}$(the expanded set of names), in order to reduce the number of candidate minimal sets. So, while in the above example we still have the same choices, since each introduces only one additional concept, if the original ontology had axioms $Feline \sqsubseteq Mammal$, $Mammal \sqsubseteq Animal$ instead of $Feline \sqsubseteq Animal$, then KB$_{sub}$=$\{Cat \sqsubseteq Feline$, $Feline \sqsubseteq Mammal,\ Mammal \sqsubseteq Animal\}$ would be eliminated since it introduced two, rather than one, additional identifier.

Turning to expansion of the vocabulary, the logical consequence approach is seriously deficient, since it is based entirely on the original set of names Ω, and therefore provides no guidelines for constructing $\Omega_\oplus$. The other two approaches partially address the issue because when choosing a subset of formulae to cover E^Ω, one is forced to use additional names appearing in them, which become part of $\Omega_\oplus$. For example, if $\mathcal{O}$= $\{Married\equiv Person\sqcap \geqslant 1\, spouse,\ Bachelor\equiv Person\sqcap\ Male\sqcap \leqslant 0\, spouse\}$, and Ω={Married, Bachelor}, then, to capture their disjointness, we will want import$_2$(Ω,$\mathcal{O}$) to contain { Married $\sqsubseteq\ \geqslant 1\, spouse$, Bachelor $\sqsubseteq\ \leqslant 0\, spouse$ }, introducing the new identifier spouse. One could also make a case that the actual definitions should be included, since users of the term should appreciate that these are defined, as opposed to primitive concepts. A compromise might be to allow for definitions with ellipses: { Married $\equiv \ldots \sqcap \geqslant 1$ spouse, Bachelor $\equiv \ldots \sqcap \leqslant 0$ spouse }.

It remains unclear however, how one would enforce by strictly logical means general policies such as “if A and B are included, then so should their least common parent in the IsA hierarchy”— a policy that seems common in many of the examples reviewed. This comment also applies to all purely logic-based specifications of importing described below. An attempt is made to address this issue in [9], using a functional approach to knowledge bases.

4.4.3 The Influence of the Importing Ontology

Both previous techniques ignore the purpose for which the names in Ω are being imported. One way to make this dependency explicit is to view importing as a ternary relationship import$_3$(Ω, KB$_{expt}$, KB$_{impt}$), where KB$_{impt}$ plays an important role as KB$_{expt}$ in determining what is to be imported. To quote from a recent paper that talks about modularization based on such principles [19],

> “ …the central requirement for a module $Q_1 \subseteq Q$ to be reused in our ontology P is that $P \cup Q_1$ should yield the same logical consequences in the vocabulary of P as $P \cup Q$ does.”

When adding the set Ω of names to be imported, this means that the minimal material to be imported is characterized not by E^Ω but by E^Ω_{impt}, specified as

$$\{\varphi\ \mid vocab(\mathrm{KB}_{impt} \cup \{\varphi\}) \cap vocab(\mathrm{KB}_{expt}) \subseteq \Omega \text{ and } \mathrm{KB}_{impt} \cup \mathrm{KB}_{sub} \models \varphi\}$$

Since the above specification allows every formula entailed by KB$_{expt}$ with vocabulary in Ω to be considered, clearly everything in import$_2$(Ω,KB$_{expt}$) (= E^Ω from Section 4.4.2) is included in E^Ω_{impt}, for any KB$_{impt}$. The question arises whether the converse also holds true; i.e., whether KB$_{impt}$ $\cup$ import$_2$(Ω, KB$_{expt}$) $\models$ E^Ω_{impt}. It turns out that a counter-example can be constructed when there is implicit information about Ω in KB$_{impt}$ $\cup$ KB$_{expt}$ that cannot be captured explicitly by the logic used in KB$_{impt}$$\cup$ KB$_{expt}$. For example, consider a Description Logic which states axioms of UML using domain, range and number restrictions on roles, but without concept disjunction/negation traditionally used to express subclass disjointness and covering. In particular, suppose this formalism uses an axiom of the form cover(A,{B,C}) to indicate that A is partitioned by subclasses B and C. Given KB$_{expt}$={cover(A,{B,C}), D $\sqsubseteq$ A}, if Ω={D,B,C}, then

$\mathsf{import}_2(\Omega,\mathrm{KB}_{expt})$ is the empty set since there is no formula *not* involving A, which captures the disjointness of B and C, or the subsumption D $\sqsubseteq$ (B $\sqcup$ C). On the other hand, if KB_{impt}=$\{$B $\sqsubseteq \bot\}$, then a sound and complete reasoner would be able to deduce from $\mathrm{KB}_{impt} \cup \mathrm{KB}_{expt}$ that D $\sqsubseteq$ C. Therefore the cover-expression itself (in the syntactic subset approach) would have to be included in $\mathsf{import}(\Omega,\mathrm{KB}_{expt},\mathrm{KB}_{impt})$.

The preceding specification of import might be considered problematic when KB_{impt} can change, because every time the importing ontology is modified, the material to be imported needs to be recomputed. So "the understanding of the meaning of terms in Ω" changes as the knowledge base KB_{impt} is built. (Note that proposals for connecting multiple ontologies using so-called "bridge rules" [11, 5] behave exactly in this way, relying on the precise content of both the importing and exporting ontology.)

A natural approach is to revert to a (modified) binary definition $\mathsf{import}_2^{e,i}(\Omega, \mathrm{KB}_{expt})$ — one which considers *all* possible importing knowledge bases. In this case, the material to be imported, KB_{sub}, once again a subset of (the logical consequences of) KB_{expt}, needs to satisfy the property that

for all $\mathrm{KB}_{impt}, \varphi$ such that $(vocab(\mathrm{KB}_{impt} \cup \{\varphi\}) \cap vocab(\mathrm{KB}_{expt})) \subseteq \Omega$,
we have that $(\mathrm{KB}_{impt} \cup \mathrm{KB}_{expt}) \models \varphi$ iff $(\mathrm{KB}_{impt} \cup \mathrm{KB}_{sub}) \models \varphi$.

4.4.4 Some Restrictions on Importing

First, it is interesting to note that a number of proposals for knowledge import *constrain the use of imported names*. For example, in the work of Navigli, Swartout *et al*, and Conesa&Olive (see Sections 4.2.2 and 4.2.4) the concepts in Ω are required to be super-classes of concepts that have been gathered by other means, since WordNet and Cyc are supposed to act as top-level, general ontologies. On the other hand, the proposals based on $\mathcal{E}$-Connections[25, 16], essentially allow an ontology $\mathcal{O}_1$ to use concepts from $\mathcal{O}_2$ (which can even be based on a different logic) as long as these imported terms are used only as role restrictions on certain specially designated roles. For example, various types of buildings may have their location restricted in spatial logic through role locatedIn. One way to capture these restrictions formally is to provide a *grammar* $\mathcal{G}_\Omega$ specifying the use of the imported names. For example, the following trivial grammar $\mathcal{G}_{winnow}$ restricts the use of imported names in TBox subsumption axioms as in Sections 4.2.2 and 4.2.4:

```
<TBox axiom> ::= <local axiom> | <connect up axiom>
<local axiom> ::= <local DL concept> ⊑ <local DL concept>
<connect up axiom> ::= <local DL concept> ⊑  <Imported identifier>
<local DL concept> ::= ...
```

Therefore, it seems reasonable to consider a version of import which takes as an additional argument a description of the restrictions of the place where imported names can appear: $\mathsf{import}_3^{G}(\Omega,\mathrm{KB}_{expt},\mathcal{G}_\Omega)$.

Second, we have assumed so far that the importing and exporting knowledge base share, if not the same vocabulary, at least the same logic. However, as illustrated in Section 4.2.3, there is a need to consider cases where the logics are different. It seems reasonable to require that the material imported, KB_{sub}, be represented in the language of the importing ontology. For example, if KB_{impt} uses Datalog to represent ontologies, while KB_{expt} is expressed in CycL, with its FOL and default rules, it does not make sense to show CycL definitions in KB_{impt}. This means that the definitions in the immediately preceding section, which consider the union of KB_{impt} and KB_{expt}or KB_{sub}, are likely to not work well. For example, if KB_{expt}=$\{$A $\sqsubseteq$ (B $\sqcup$ D) $\}$, then an importing ontology that uses only the $\mathcal{AL}$ DL cannot capture $\mathsf{import}_2^{e,i}(\{$A,B,D,$\}$, $\mathrm{KB}_{expt})$ because in case $\mathrm{KB}_{impt} \models$ B $\sqsubseteq$ D, it needs to be able to deduce A $\sqsubseteq$ D, yet this is not entailed by KB_{expt}, and hence cannot be imported. As in Section 4.2.3, the more reasonable solution is to approximate $\mathsf{import}_2(\Omega,\mathrm{KB}_{expt})$ in the logic of KB_{impt}. One can actually fold this into the framework of $\mathsf{import}_3^G(\Omega,\mathrm{KB}_{expt},\mathcal{G}_\Omega)$, by using as $\mathcal{G}_\Omega$ the grammar describing the syntax of the logic used in KB_{impt}.

It is useful to note that by limiting the use of the imported names, both of the above cases may enable *MinPrinciple* to be applied more successfully because not all formulas in E^Ω can be asked as valid queries or lead to useful new inferences. For example, even if $\mathrm{KB}_{expt} \models N_1 \sqsubseteq (N_2 \sqcup N_3)$, and $\{N_1, N_2, N_3\} \subseteq \Omega$, $\mathsf{import}_3^G(\Omega,\mathrm{KB}_{expt},\mathcal{G}_\Omega)$ need not entail $N_1 \sqsubseteq (N_2 \sqcup N_3)$ when KB_{impt} is a weak DL where one can only assert or question subsumptions between atomic concept names, and KB_{expt} does not support negation.

4.4.5 On Some Existing Proposals for Importing

Most of the chapters in this book deal with modularizing ontologies. Only two, [19] and [8], explicitly consider the list of names to be imported, and how this influences the subset of the ontology that needs to be imported. Using the notions introduced in the preceding subsections, we can now point out the position staked out by these two proposals.

Cuenca Grau et al [19] propose Ω-modules (called "S-modules" in the paper) as $\mathsf{import}_2^{e,i}$, for the case of possibly differing logics, using the syntactic subset approach, and applying *MinPrinciple* to the set of axioms returned. We refer the reader to [20] for additional details of this approach.

Borgida [8] proposes a definition of import that is like import_3^G, allowing both restricted use of imported names and different logics; it uses the expanded axioms approach, and applies *MinPrinciple* to both the *size* of the expanded set of names and the set of axioms imported.

One could try to view several of the other entries in Part III (e.g., Chapters 13 and 14) in light of import_3^G: the grammar $\mathcal{G}_\Omega$ specifies the *precise* content of KB_{impt} and how it uses names in Ω. More importantly, the axioms in so-called "bridge rules" should not be interpreted as normal subsumptions, but have instead a special semantics via mapping functions connecting the ontology domains.

4.5 On the Notion of "Expressive Power"

Given the variety of approaches to defining the notion of importing and its formal semantics, it is not surprising that one would be interested in comparing them. For example, [25, 16, 30, 5, 6] contain (sometimes contradictory) statements concerning the expressive power of formalisms: "Formalism X is more expressive than Y", "X cannot express particular concept/situation β".

The purpose of this section is, first, to alert the community to the need to proceed carefully in this regard: just because the obvious way of stating β in X does not have the right consequences does not mean that it cannot be expressed in formalism X. To make a convincing case, one needs to start with an *appropriate formal definition(s) for the notion of expressive power*, and then provide a mathematical proof that there is no way that β can be expressed.

A second goal of this short section is then to provide some examples of how expressive power has been defined in prior research, in areas such as databases and description logics, thus possibly providing a model for how research on expressiveness of import, or more generally module connection mechanisms might proceed. Regretfully, we provide no new formal results here.

Finally, in Section 4.5.3, we briefly look at some of the claims about formal expressive power of module/ontology interconnections which have appeared in the literature.

4.5.1 Expressive Power of Query Languages

In the field of databases, the notion of "a query language being as/more expressive than another" has been crucial since the advent of declarative query languages. In particular, right at the beginning, Codd showed that relational algebra and relational calculus were equally expressive as query languages for relational databases. To show that query language $\mathcal{L}_1$ is *as expressive as* $\mathcal{L}_2$, one views a query Q as a function f_Q from database instances to answers (sets of tuples), and then shows that for every query Q in $\mathcal{L}_1$ there is a query S_Q in $\mathcal{L}_2$ which computes the same function. This is usually done by providing a translation τ which takes as input queries Q in $\mathcal{L}_1$ and produces a query τ(Q) in $\mathcal{L}_2$. For a satisfying state of affairs, τ is usually defined by structural recursion, and is therefore constructive.

To show that language $\mathcal{L}_1$ is *more expressive* than $\mathcal{L}_2$ one must therefore prove that there *can be no query written in* $\mathcal{L}_2$ that expresses the function described by some query in $\mathcal{L}_1$. An example of such a direct proof is the result that transitive closure cannot be expressed in relational algebra [2].

More often, this is accomplished via complexity theory: showing that the queries in $\mathcal{L}_2$ compute functions in some complexity class strictly contained in the class of functions computed by $\mathcal{L}_1$.

4.5.2 Expressive Power of Description Logic

In establishing an upper bound on the expressiveness of current description logics, [7] follows a function-based approach by equating concepts C and roles r to unary and binary functions $\lambda x.C(x)$ and $\lambda x, y.r(x, y)$, and then showing that these functions could be expressed in First Order Logic with at most 3 variable symbols, counting quantifiers and fix-point operator. It is important to note that while complexity results concerning the family of such functions that can be captured by some DL can still be used to distinguish the expressive ability of description logic, complexity results concerning *subsumption* cannot: these deal with a different issue, sometime tied to the brevity with which concepts can be expressed.

Baader [4] undertook a direct study of the comparative expressive power of Description Logics. For example, if a DL has number restrictions but no constant for the bottom/inconsistent concept $\perp$, then one wants to say that adding $\perp$ does not increase expressive power since all its occurrences can be replaced by $\geqslant 1\, p \sqcap \leqslant 0\, p$, which has the same denotation. This kind of replacement can be accomplished by a translation function τ of a similar nature to the one introduced for query languages.

The novelty here is that one is interested not just in the ability to express a particular concept, but in the ontologies/TBoxes built with them, from which one can make deductions. In this respect, one can compare two description logics, $\mathcal{DL}_1$ and $\mathcal{DL}_2$, on strictly model-theoretic grounds or on the basis of deductions supported. In particular, one can say $\mathcal{DL}_2$ is as expressive as $\mathcal{DL}_1$ iff for every TBox $\mathcal{T}$ of $\mathcal{DL}_1$ there is a corresponding TBox $\tau(\mathcal{T})$ in $\mathcal{DL}_2$ such that either

- $\mathcal{T}$ and $\tau(\mathcal{T})$ have the same models[4];

or

- $\mathcal{T}$ and $\tau(\mathcal{T})$ have the same logical consequences; i.e.,

$$\mathcal{T} \models \alpha \sqsubseteq \beta \text{ iff } \tau(\mathcal{T}) \models \tau(\alpha) \sqsubseteq \tau(\beta)$$

These two definitions are not identical. For example, consider $\mathcal{DL}_1$, which supports only concept constructors $\{\sqcap, \forall\}$, while $\mathcal{DL}_2$also supports the declaration of transitive roles in the TBox. In this particular case, there are no new subsumptions that can be deduced as a result of a role being declared transitive. (Note that this intuitively obvious statement needs to be backed up by a tedious inductive proof on the structure of all possible subsumptions judgments.) However, a $\mathcal{DL}_2$ontology $\mathcal{T}$= { A $\sqsubseteq \exists p.B$, transitive(q) } has only models that have transitive relationships as denotations of q, and there seems to be no way to achieve this in $\mathcal{DL}_1$. (Again, this intuitive statement needs to be converted into an actual proof about the absence of a possible translation function — a proof that is much easier for $\mathcal{T}_0$ = {transitive(q)}.)

What is happening here is that the TBox language can make assertions (about transitivity) that result in model theoretic differences, but these models cannot be distinguished using only deductions concerning subsumptions. (Were one allowed to ask whether $\mathcal{T} \models \mathit{transitive}(p)$, the situation would be different.)

A real-world case of this occurs in [12], where the constraint that certain classes of individuals must have unique combinations of values for so-called "key" features {q_1,q_2,...} is shown to have no influence on the subsumptions that can be concluded.

[4] To be well-defined, this requires that the two DLs have the same notion of interpretation structure.

Finally, the following example from [4] shows that the issue is also more complex concerning the translation function τ: Suppose that $\mathcal{DL}_2$ supports TBoxes which allow only concept definitions, of the form $A \equiv \alpha$, where A is a concept name, while $\mathcal{DL}_1$ supports TBoxes which allow both concept definitions $A \equiv \alpha$ and subsumption axioms of the form $A \sqsubseteq \beta$. There is a well-known technique for encoding $\mathcal{DL}_1$ TBoxes in $\mathcal{DL}_2$ TBoxes, illustrated by the following example: the axiom $\mathsf{Dog} \sqsubseteq \mathsf{Pet}$ is replaced by $\mathsf{Dog} \equiv \mathsf{Dogness} \sqcap \mathsf{Pet}$, where $\mathsf{Dogness}$ is a new primitive concept identifier. There is then an intuitive sense in which $\mathcal{DL}_1$ and $\mathcal{DL}_2$ are equally expressive. However, a translated $\mathcal{DL}_1$ ontology $\tau(\mathcal{T})$ then contains additional symbols, thus both entailing new subsumptions (e.g., $\mathsf{dog} \sqsubseteq \mathsf{Dogness}$), and having different interpretations. Baader [4] appropriately formulates the question of model-theory based expressive power in the context of more complex translations (essentially computable translation functions), and studies the model-theoretic definition.

4.5.3 Expressivity of Connected Logics

Bao et al [6] undertake a formal investigation of expressive power for the family of import approaches dealing with many ontologies, which use as semantic interpretation a structure containing multiple classical "local worlds", each interpreting some ontology that may import or export knowledge, and connecting their domains with so-called domain mappings (Chapters 13, 14). The paper [6] paper provides a list of "expressivity criteria", such as "Can the language express constraints of the form $A \sqsubseteq \exists R.C$ where C and R are imported from an outside ontology?". Such criteria are quite syntactic (and superficially easy to answer), in contrast to something like "Can the language express constraints that ensure that in every model, the interpretation of A is contained in the appropriate set of objects described in terms of the interpretations of R and C." The answer to the second question is much harder (especially if it is going to be negative). Yet, as illustrated by the examples in Section 4.5.2, this seems a much more relevant issue, if one is going to consider expressive power. As it happens, [6], does provide a common model theory that underlies the formalisms—the notion of Abstract Modular Ontology. One could therefore proceed along the lines described in the previous subsection for simple description logics.

Chapter 11 also provides a comparison between these formalisms, this time based on their mapping to Distributed First Order Logic (DFOL). This mapping is useful for the stated purpose of comparing assumptions, and differences in interpretation for the notion of "ontology mapping". It should not however be used directly to compare expressive power (by comparing sets of DFOL formulas covered) because that depends on the particular mapping to DFOL chosen. One proper approach would be to find a way to associate functions on models or individuals with DFOL theories.

Similarly, statements of the form "Formalism X cannot capture example e/phenomenon F" (e.g., "It is inconsistent to say that $\mathsf{Penguin}$ in $\mathcal{O}_2$ is subsumed by Bird and $\neg\mathsf{Bird}$ in $\mathcal{O}_1$") can only be established by formal proof, once one chooses a formal definition of how one can say things—a nontrivial task in the presence of domain mappings, etc. Moreover, as shown in [4], such impossibility proofs are liable to be quite difficult.

4.6 Conclusions

The aim of this chapter has been to gain a better understanding of the notion of connecting ontologies through a mechanism that allows knowledge from one ontology to be imported into another. Rather than addressing directly the question "What is a module?", it focuses on "What do I need from ontology $\mathcal{T}$ if I want to understands notions $\{C_1, C_2, \ldots\}$, which are described there?". The result is therefore not an *a priori* modularization of ontology $\mathcal{T}$ (based on its internal structure) — as studied in [17, 29], for example, but a modularization *relative to the needs of understanding the names in set Ω* — what one might term "demand-based module extraction", as studied in [19, 8]. This topic is closely related, at this high level, with other entries in Part III, which deal with more precise ways to connect disparate ontologies, but which are distinguished by a more complex formal semantics, involving multiple local models.

We started by surveying a number of proposals for importing knowledge that had been used in Databases and Artificial Intelligence. From this, we extracted a few common features, such as soundness, the need to possibly import additional identifiers to help explain the meaning of those in Ω, and the desire to minimize the material imported (both the set of extra names and the set of axioms). We then looked at a spectrum of options available in the logical specification of import(Ω,KB_{expt}). These included the question whether the axioms imported are to form a subset of the actual axioms in KB_{expt}, or whether they should only be entailed by it. The most interesting aspect here was the fact that all the proposals reviewed in Section 4.2 actually imported a subset not of the original knowledge base, but of an *enlargement* of it with a carefully circumscribed set of axioms. Moreover, we pointed out that often this enlargement corresponded to "lemmas" that followed from KB_{expt} but which would not need to be explained to humans because their deduction seemed trivial. Next, we looked at different ways in which import could be specified logically: (i) by using *a priori* modularizations of KB_{expt}; (ii) by considering only the axioms needed to prove the formulas entailed by KB_{expt}; (iii) by also considering the importing ontology; (iv) by circumscribing the class of importing ontologies to limit the use of Ω, or the logic used. This allowed us to see the precise differences between two earlier proposals for import [19, 8]. Because of the plethora of choices available (both here and in other parts of the book), there is a temptation to make statements that certain approaches are more expressive than others or cannot express certain situations. For this reason, we ended with a review of (what we believe are) appropriate grounds on which "more expressive than/can express" has been treated in the field of database query languages and ordinary Description Logics, and then briefly commented on current efforts to compare proposals in this book.

References

1. `http://www.loa-cnr.it/medicine/`
2. Aho, A.V., Ullman, J.D.: The Universality of Data Retrieval Languages. In: POPL 1979, pp. 110–120 (1979)

3. Baader, F., Ksters, R., Molitor, R.: Rewriting Concepts Using Terminologies. In: KR 2000, pp. 297–308 (2000)
4. Baader, F.: A Formal Definition for the Expressive Power of Terminological Knowledge Representation Languages. J. Log. Comput. 6(1), 33–54 (1996)
5. Bao, J., Caragea, D., Honavar, V.: On the Semantics of Linking and Importing in Modular Ontologies. In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L.M. (eds.) ISWC 2006. LNCS, vol. 4273, pp. 72–86. Springer, Heidelberg (2006)
6. Bao, J., Caragea, D., Honavar, V.: Modular Ontologies - A Formal Investigation of Semantics and Expressivity. In: Mizoguchi, R., Shi, Z.-Z., Giunchiglia, F. (eds.) ASWC 2006. LNCS, vol. 4185, pp. 616–631. Springer, Heidelberg (2006)
7. Borgida, A.: On the Relative Expressiveness of Description Logics and Predicate Logics. Artif. Intell. 82(1-2), 353–367 (1996)
8. Borgida, A.: On Importing Knowledge from DL Ontologies: intuitions and problems. In: DL 2007, CEUR On-line Workshop Proceedings, vol. 250 (2007)
9. Borgida, A., Giunchiglia, F.: Importing from Functional Knowledge Bases – A Preview. In: WOMO 2007: CEUR On-line Workshop, vol. 315 (2007)
10. Borgida, A., Franconi, E., Horrocks, I.: Explaining ALC Subsumption. In: ECAI 2000, pp. 209–213 (2000)
11. Borgida, A., Serafini, L.: Distributed Description Logics: Assimilating Information from Peer Sources. J. Data Semantics 1, 153–184 (2003)
12. Calvanese, D., Lenzerini, M., Nardi, D.: Unifying Class-Based Representation Formalisms. J. Artif. Intell. Res. (JAIR) 11, 199–240 (1999)
13. Conesa, J., Olivé, A.: A Method for Pruning Ontologies in the Development of Conceptual Schemas of Information Systems. J. Data Semantics V, 64–90 (2006)
14. Conesa, J., Olive, A.: Pruning Ontologies in the Development of Conceptual Schemas of Information Systems. In: Atzeni, P., Chu, W., Lu, H., Zhou, S., Ling, T.-W. (eds.) ER 2004. LNCS, vol. 3288, pp. 122–135. Springer, Heidelberg (2004)
15. Gangemi, A., Pisanelli, D.M., Steve, G.: An Overview of the ONIONS Project: Applying Ontologies to the Integration of Medical Terminologies. Data Knowl. Eng. 31(2), 183–220 (1999)
16. Grau, B.C., Parsia, B., Sirin, E.: Working with Multiple Ontologies on the Semantic Web. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 620–634. Springer, Heidelberg (2004)
17. Grau, B.C., Parsia, B., Sirin, E., Kalyanpur, A.: Modularity and Web Ontologies. In: KR 2006, pp. 198–209 (2006)
18. Grau, B.C., Horrocks, I., Kazakov, Y., Sattler, U.: A Logical Framework for Modularity of Ontologies. In: IJCAI 2007, pp. 298–303 (2007)
19. Grau, B.C., Horrocks, I., Kazakov, Y., Sattler, U.: Just the Right Amount: Extracting Modules from Ontologies. In: WWW 2007, pp. 717–726 (2007)
20. Grau, B.C., Horrocks, I., Kazakov, Y., Sattler, U.: Modular Reuse of Ontologies: Theory and Practice. JAIR 31, 273–318 (2008)
21. Fikes, R., Farquhar, A., Rice, J.: Tools for Assembling Modular Ontologies in Ontolingua. In: AAAI/IAAI 1997, pp. 436–441 (1997)
22. Guha, R.V., Lenat, D.B.: Enabling Agents to Work Together. CACM 37(7), 126–142 (1994)
23. Gangemi, A., Guarino, N., Masolo, C., Oltramari, A., Schneider, L.: Sweetening Ontologies with DOLCE. In: Gómez-Pérez, A., Benjamins, V.R. (eds.) EKAW 2002. LNCS, vol. 2473, pp. 166–181. Springer, Heidelberg (2002)

24. Herzig, A., Varzinczak, I.J.: A Modularity Approach for a Fragment of ALC. In: Fisher, M., van der Hoek, W., Konev, B., Lisitsa, A. (eds.) JELIA 2006. LNCS, vol. 4160, pp. 216–228. Springer, Heidelberg (2006)
25. Kutz, O., Lutz, C., Wolter, F., Zakharyaschev, M.: E-connections of abstract description systems. Artif. Intell. 156(1), 1–73 (2004)
26. McGuinness, D.L., Borgida, A.: Explaining Subsumption in Description Logics. In: IJCAI 1995, pp. 816–821 (1995)
27. Navigli, R.: Extending, Pruning and Trimming General Purpose Ontologies. In: IEEE SMC 2002 (2002)
28. Peterson, B.J., Andersen, W.A., Engel, J.: Knowledge Bus: Generating Application-focused Databases from Large Ontologies. In: KRDB 1998, pp. 2.1–2.10 (1998)
29. Seidenberg, J., Rector, A.L.: Web ontology segmentation: analysis, classification and use. In: WWW 2006, pp. 13–22 (2006)
30. Serafini, L., Borgida, A., Tamilin, A.: Aspects of Distributed and Modular Ontology Reasoning. In: IJCAI 2005, pp. 570–575 (2005)
31. Stuckenschmidt, H., Klein, M.C.A.: Integrity and Change in Modular Ontologies. In: IJCAI 2003, pp. 900–908 (2003)
32. Swartout, W.R., Tatil, R., Knight, K., Russ, T.: Toward Distributed use of Large-Scale Ontologies. In: KAW 1996 (1996)
33. Wouters, C., Dillon, T.S., Rahayu, J.W., Chang, E.: A Practical Walkthrough of the Ontology Derivation Rules. In: Hameurlain, A., Cicchetti, R., Traunmüller, R. (eds.) DEXA 2002. LNCS, vol. 2453, pp. 259–268. Springer, Heidelberg (2002)

5

Modularity in Databases

Christine Parent[1], Stefano Spaccapietra[2], and Esteban Zimányi[3]

[1] HEC ISI, Université de Lausanne, Switzerland
christine.parent@epfl.ch
[2] Database Laboratory, Ecole Polytechnique Fédérale de Lausanne, Switzerland
stefano.spaccapietra@epfl.ch
[3] Department of Computer and Decision Engineering (CoDE), Université Libre de Bruxelles, Belgium
ezimanyi@ulb.ac.be

Summary. Modularization can be sought for as a technique to provide context-dependent perspectives over a given shared information repository. This chapter presents an approach to database modularization where the modules represent application-specific perspectives over the shared database. The approach is meant to support the creation/definition of the modules as part of the conceptual schema definition process, that is to say the modules and the database they are a subset of are simultaneously defined. This is similar to Cyc's approach to ontological microtheories definition. The chapter develops both intuitive and formal definition of the proposed approach. It also shows the basics of how the modules are used by user transactions and of how the overall multiperception database can be implemented on a commercial database management system.

5.1 Introduction

A database stores a *representation* of the part of the real world that is of interest for a set of applications. Usually, information requirements vary from one application to another and call for different representations of the real world. For example, given a database describing vineyards, one application may focus on production data (e.g., which wines, which qualities and quantities) while another application focuses on cultivation aspects (e.g., which plants, fertilizers, harvesting techniques). Traditional database models[1] poorly comply with such situations as they do not explicitly support the definition of several representations for the same real-world phenomenon. Database designers have the choice between two unsatisfactory solutions. One is to define two tables (assuming a relational database) with different names (e.g., VineyardProduction and VineyardCultivation), each one with its attributes. To maintain the consistency of two instances (one in each table) describing the same vineyard,

[1] This chapter employs the database terminology. The term "model" means the schema language that allows designers to define the schema (description) of their database.

integrity constraints have to be defined to force attributes shared by the two focuses (e.g., an attribute holding the surface of the vineyard) to have the same value in the two instances. Querying such a database is uneasy as the user has to pay attention on which tables to query (one or the other or both). The second solution, more frequently used, is to merge all the desired representations into a common unique representation, the schema of the database. The *view* mechanism is then used to construct alternate representations that differ from those stored. In the example, the designer would first define in the schema a unique Vineyard table with all attributes (those common to the two focuses as well as those relevant for only one focus). Second, the designer would define two views, a VineyardProduction view and VineyardCultivation view, where each view extracts from the Vineyard table the attributes relevant to the targeted application. Notice that this second solution can only cope with compatible representations, where differences can be readily adjusted using the facilities of the manipulation language (e.g., SQL). Unfortunately, there are situations where differences between representation requirements go beyond the restructuring capabilities of SQL. For example, two applications may need the same information, e.g., an attribute A, but in incompatible formats, which would typically lead current systems to define in the schema table two attributes with different names, A1 and A2, and have each application view separately recover the A attribute from the base table attributes A1 and A2. The drawback of this solution is that the system ignores that A1 and A2 represent the same information and is consequently not in a position to guarantee the consistency of the two representations. Situations of this kind typically arise when creating a federated database out of a set of existing databases that represent the same phenomena in different ways, or in geographic applications that need to store the spatial extent of objects at different spatial resolutions. Current database management systems (DBMS) and geographic information systems (GIS) provide a few tools for explicitly supporting *multiple representations*. DBMS use generalization/specialization links to provide users with several representations of the same real-world entity with different levels of details. Some GIS allow storing several geometries for each spatial object.

Thanks to the flexibility it supports and to its relative simplicity, the view mechanism has become extremely popular with database users and designers, and has also influenced work on ontology modularization (see Part II of this book). This is despite the fact that views do not provide a complete solution to the problem. Inherently to the approach, each view is a single virtual table whose instances are derived from the stored database. Most applications need instead access to a virtual database holding (as any database does) sets of interrelated data from different tables. In current relational technology, these applications need to define one view per table they need, make sure they do not loose the external key defining the connections between the tables (e.g., use precomputed joins rather than the original tables, as referential integrity between the views is not supported), and acquire access rights to all their view tables. This is not that simple and risks of inconsistency are high. The idea of a virtual database has been initially proposed in the 1960s under the term subschema and was implemented in legacy systems such as Codasyl DBMSs. Unfortunately (for application developers) it was discarded once the view mechanism was invented for the benefit of DBMS developers.

As the name says, *subschemas* relied on the idea that each application needs only a subset of the database. In this chapter we propose an approach to resume this idea, while making it more general. Instead of associating an application with a subschema we make it possible for each application to have its own schema (and database) while keeping the correlations with the schemas of the other applications. All application schemas (and databases) are stored within a single database, which we call a "multiperception" database. We say each application has its own "perception" of the multiperception database, and automatically gets from the DBMS the data corresponding to its perception. Equally correct would be to say that we change a traditional database into a multiperception database, and then each application can have its subschema corresponding to its perception of the multiperception database. To be precise, a *perception* in our approach is defined as the set of representations of all objects and links corresponding to a specific usage of the database. For instance, in a geographic database used for producing maps of a country at two different scales, say 1:20'000 for hikers and 1:300'000 for car drivers, it would be useful to group in a first perception all the 1:20'000 representations and in another one all the 1:300'000 representations. Users of this database could then open the database with the perception they need and get the corresponding homogeneous set of representations, i.e., a virtual, consistent single perception database.

The multiperception idea provides a possible approach for modularization. Given a database DB that one wants to modularize into modules $M_1, \ldots, M_n$, the process simply requires to define $M_1, \ldots, M_n$ as the desired perceptions and then tag each element of the database with the perception(s) it belongs to. This applies whatever the chosen technique for splitting the database into modules is. The approach is also independent of the data model (relational, UML, . . .) used by the database designer, and can therefore also apply to ontologies. Indeed, a very similar approach is the one followed by Cyc to define *microtheories* within an ontology [5].

However, implementing a multiperception approach is not just a matter of using a set of tags. Its full specification requires the definition of tagging consistency rules to guarantee that each perception defines a coherent database, the definition of how a multiperception database can be manipulated by applications that may or may not want to share information, and the definition of how interperception processes can be supported, in particular with interperception links.

This chapter describes the capabilities we have defined as an answer to this need for multiperception data. These capabilities are embedded into a conceptual data model, Mads, that we had developed for classic as well as geographic and temporal databases. The Mads mechanism for multiple perceptions and representations allows any kind of element of the database to have several representations, and allows each user to get his/her own perception of the database. In the following we use Mads terminology, in particular the term "perception", which the reader can read as "module".

Mads is primarily intended for database designers, i.e., persons in charge of specifying the schema of a database in response to user/application requirements. Thus, it is a *conceptual* model: It enables a direct mapping between the perceived world and its representation. Using Mads, designers can focus exclusively on the requirements of their applications without having to care about implementation concerns. Mads is complemented with data manipulation languages that allow users to specify queries and updates at the conceptual level too. A set of tools developed during the European project MurMur automatically implements the conceptual specifications (schema or query) onto a DBMS or GIS [11].

A Mads database is defined from the very beginning as containing a set of objects and relationships that may be shared by several perceptions, each object or relationship being possibly perceived in a different way for each perception. Using ontology terminology we would say that the Mads perceptions share a common interpretation domain: There is a unique global set of object identifiers (oid) and a unique global set of relationship identifiers (rid) which are common to all perceptions. This approach is different from Cyc where assertions of two different microtheories (which are not a super- and its sub-microtheory) are always independent. The basic principles of Mads multiple perceptions and representations are: 1) Any database element, be it composite (e.g., an object type) or atomic (e.g., a simple attribute) may have various representations, one per perception. Any two perceptions may share any kind of element of the database. 2) Two objects belonging to two different perceptions may be linked by a binary relationship or by a multi-instantiation link (is-a or overlap link). These points are developed in the following sections.

Section 5.2 sets the Mads framework by giving an overview of the characteristics of the Mads data model, excluding the perceptions and representations aspect. Section 5.3 defines the various kinds of perceptions, and shows how to design and use perceptions. Sections 5.4 and 5.6 present, respectively, the various kinds of inter-perception links and the dependencies between perceptions that these links generate. Section 5.8 describes how to implement the Mads model in the relational model, while Section 5.5 gives a formal definition of the Mads model with the perception dimension. Section 5.9 compares the Mads approach with the ones of modular ontologies. Finally, Section 5.10 concludes this chapter and points to future research.

5.2 An Overview of the Mads Data Model

This section briefly presents the thematic, spatial, and temporal modeling dimensions of the Mads data model. All three dimensions can provide criteria for modularization. For example, spatial resolution is frequently used by geographic data providers to build modules that target production of maps at some specific scale. Maps at different levels of detail require data representations tailored to a specific user population: pedestrians, hikers, cyclists, car drivers, truck drivers, trip planners, etc. Similarly, temporal features may be used to identify modules whose data is relevant for a specific timeframe, e..g. the enterprise financial data for this year, for the year before, etc. For sake of brevity, the discussion of the perception dimension (Sections 5.3 to 5.6) does not explicitly address its relationships to spatio-temporal features. They are addressed implicitly by considering them as included in the generic structural concept of attribute. In particular, we only provide a formal definition of structural

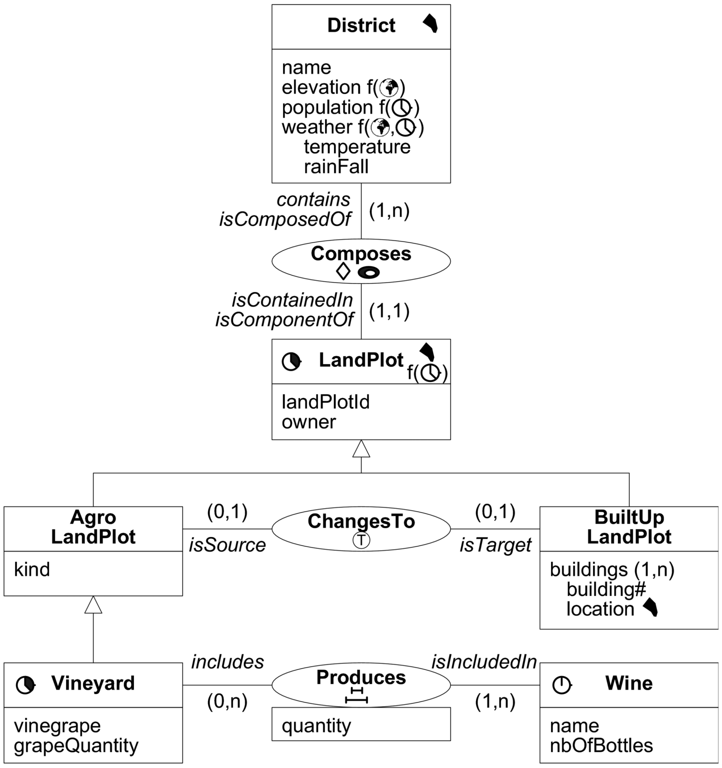


Fig. 5.1. The Mads schema of a spatio-temporal database

constructs for a multiperception database. However, the running example used in this chapter uses data with spatial and temporal features.

Readers interested in more detailed presentations of the MADS concepts and rules, including the formal definition of the model, may refer to [10, 12, 2]. The perception and representation characteristics are described and discussed in Sections 5.3 to 5.6. Unless the contrary is explicitly stated, examples in this section refer to Fig. 5.1, which describes districts that are composed of land plots, where some land plots may be built up while others are agricultural and, in particular, vineyards.

5.2.1 Structural Modeling

We first give an informal presentation. A formal one follows later. Mads structural dimension describes the chosen data structures based on well-known features such as objects and object types, relationships and relationship types, attributes, and methods[2]. Objects and relationships have a system-defined identity, called oid for objects and rid for relationships. Both objects and relationships may bear attributes.

[2] For space reasons, we do not provide in this paper a detailed discussion about methods in the Mads model.

Attributes may be mono-valued or multivalued, simple or complex (i.e., composed of other attributes), optional or mandatory, and may be derived (i.e., their value is computed from the values of other attributes). Referring to the example in Fig. 5.1, the attributes weather of District and buildings of BuiltUpLandPlot are both complex attributes, while all other attributes are simple. The buildings attribute is multivalued, as shown by the (1,n) notation following the attribute name: it describes the set of buildings located within the land plot, giving for each building its number and its spatial extent. All other attributes are monovalued, with default cardinality (1,1) not shown in the figure.

Semantic data models usually provide the capability to link objects through various types of relationships, each one holding a specific semantics. Mads separates the definition of relationships into two facets: (1) its structure (i.e., the roles linking object types and the relationship attributes, if any), (2) its semantics. Each relationship type may bear zero, one, or several specific semantics. Mads supports aggregation, generation, transition, topological, synchronization, and inter-representation semantics for the relationships. Aggregation (identified by the ◇ icon) is the most common one: It defines mereological (also termed component or part-of) semantics. An example is the Composes relationship. Generation relationships record that target objects have been generated by source objects. Transition semantics expresses that an object in a source object type has evolved to a new state that causes it to be instantiated in another target object type. For example, in the schema diagram of Fig. 5.1 the ChangesTo relationship type holds transition semantics (denoted by the Ⓣ icon) expressing that an agricultural land plot may become a built-up land plot. Instances of this relationship type record such transitions.

By definition, transition relationships link two instances of the same object in different states. The fact that an object may have two (or more) instances in different object types[3] is known as *multi-instantiation*. In most semantic data models multi-instantiation is supported through *is-a links*, which by definition relate two instances (one generic, one specific) of the same object (or relationship). Mads adds a complementary kind of multi-instantiation link between either object or relationship types: The *overlap link*. Overlap links are binary links expressing that the two linked object (or relationship) types may contain instances sharing the same identity. They have a less constraining semantics than the inclusion semantics of the is-a link. Overlapping is implicit between two types that share a common subtype. Otherwise it has to be explicitly defined as in databases, contrarily to description logics, two object (or relationship) types that are not related by multi-instantiation links always hold disjoint sets of instances. In other words, they cannot contain two object (or relationship) instances sharing the same oid (or rid). For example, Fig. 5.1 shows that District, LandPlot, and Wine are three disjoint object types. Conversely, Vineyard, AgroLandPlot, LandPlot, and BuiltUpLandPlot form a network of overlapping types: As the ChangesTo transition relationship type implicitly defines an

[3] In the case of temporal object types, the object instances linked by a transition relationship may be no longer active. Indeed, disabled instances of temporal types are kept in the database as long as they are needed by the applications.

overlap link between AgroLandPlot and BuiltUpLandPlot, a LandPlot object can have instances in any of the four object types.

Multi-instantiation in Mads is by default dynamic: Any object (or relationship) may acquire new instantiations or loose existing instantiations in any of the object (or relationship) types connected by the network of multi-instantiation links it belongs to. This is the case for the ChangesTo relationship. However, database designers may use explicit integrity constraints to constrain multi-instantiation within a set of related types to be static. In this case, an object or relationship, once initially created as instance of one or more types in the constrained set, cannot change its membership, i.e., it cannot acquire a new instantiation nor loose any but all existing ones (which means the object is deleted from the database).

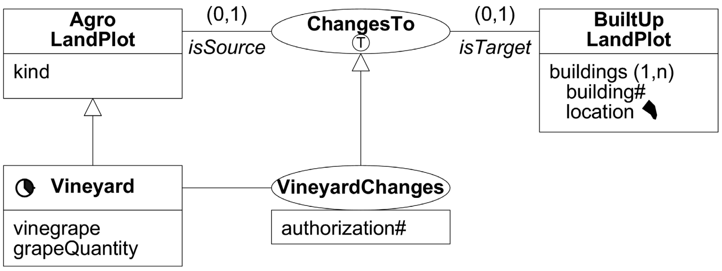


Fig. 5.2. A relationship subtype refining a role to link a subtype of the original object type

Mads supports is-a links for object and relationship types with inheritance and possibly refinement or redefinition. Such capability is needed for full flexibility in defining spatial and temporal features of subtypes (given that these features are conveyed by attributes with a fixed name). For example, in the Vineyard object type the lifecycle is redefined: It contains a time interval describing when the vineyard was productive, instead of the time interval describing when the land plot was created and deleted. An example of refinement is given in Fig. 5.2, which specifies the VineyardChanges relationship type as a subtype of the ChangesTo transition relationship. The isSource role of the relationship type is refined to link only instances of AgroLandPlot that are instances of Vineyard too. This expresses that transitions of vineyards to built-up land plots are subject to an authorization, whose number is stored in the authorization# attribute.

5.2.2 Formal Definition of Structural Constructs

For the reader unfamiliar with data modeling concepts, this section provides formal definitions for the main structural constructs of the Mads model. Namely, for sake of simplicity, we leave out the spatial and temporal dimensions. In the structural dimension, we only include the concepts that exist in most Entity Relationship data models. We do not include multiassociations, relationship semantics, complex, multivalued, and optional attributes, weak object types, and methods. Let us call SimpleMads this subset of the Mads model that we formalize below. The interested reader can refer to [10] for a full and formal description of the additional capabilities. A formal definition of the temporal dimension in the context of description logics, with its associated temporal constraints and all inferred reasoning, has been presented in [2]. Below we also leave out the perception dimension. The formal definitions of multiperception schema, multiperception database and perception are given in the following sections.

Definition 1 (SimpleMads schema without perceptions)
A SimpleMads schema without perceptions is a tuple: $\Sigma = (\mathcal{L}, \text{REL}, \text{ATT}, \text{CARD}, \text{ISA}, \text{OVLP}, \text{KEY})$, such that:

- $\mathcal{L}$ is a finite alphabet partitioned into the sets: $\mathcal{O}$ (*object type* symbols), $\mathcal{R}$ (*relationship type* symbols), $\mathcal{A}$ (*attribute* symbols), $\mathcal{U}$ (*role* symbols), and $\mathcal{D}$ (*domain* symbols).
- REL (relationships) is a total function that maps a relationship type symbol R in $\mathcal{R}$ to an $\mathcal{U}$-labeled tuple over $\mathcal{O}$, $\text{REL}(R) = \langle U_1 : O_1, \ldots, U_k : O_k \rangle$, where $k \geq 2$ is the *arity* of R.
- ATT (attributes) is a partial function that maps an object or relationship type symbol X in $\mathcal{O} \cup \mathcal{R}$ to an $\mathcal{A}$-labeled tuple over $\mathcal{D}$, $\text{ATT}(X) = \langle A_1 : D_1, \ldots, A_h : D_h \rangle$.
- CARD (cardinalities) is a partial function $\mathcal{O} \times \mathcal{R} \times \mathcal{U} \to \mathbb{N} \times (\mathbb{N} \cup \{\infty\})$ that defines cardinality constraints associated to the roles of the relationship types. For a relationship type R such that $\text{REL}(R) = \langle U_1 : O_1, \ldots, U_k : O_k \rangle$, we use $\text{CMIN}(O_i, R, U_i)$ and $\text{CMAX}(O_i, R, U_i)$ to denote the first and second component of CARD.
- ISA (is-a links) is a transitive binary relation $\text{ISA} \subseteq (\mathcal{O} \times \mathcal{O}) \cup (\mathcal{R} \times \mathcal{R})$ that defines is-a links for object and relationship types.
- OVLP (overlap links) is a symmetric binary relation $\text{OVLP} \subseteq (\mathcal{O} \times \mathcal{O}) \cup (\mathcal{R} \times \mathcal{R})$ that defines overlapping links between object or relationship types.
- KEY is a binary relation, $\text{KEY} \subseteq (\mathcal{O} \cup \mathcal{R}) \times 2^{\mathcal{A}}$, which associates to each object and relationship type symbol a set of keys, each key being composed of a set of attributes of the object or relationship type. $\diamond$

The model-theoretic semantics associated with the SimpleMads model without the perception dimension is given next.

Definition 2 (Database state of a SimpleMads schema without perceptions)
Let Σ be a SimpleMads schema without perceptions. A *database state* for the schema Σ is a tuple $\mathcal{B} = (\Delta^{\mathcal{B}}_{\mathcal{O}} \cup \Delta^{\mathcal{B}}_{\mathcal{R}} \cup \Delta^{\mathcal{B}}_{D}, \cdot^{\mathcal{B}})$, such that: the three sets $\Delta^{\mathcal{B}}_{\mathcal{O}}$, $\Delta^{\mathcal{B}}_{\mathcal{R}}$, and $\Delta^{\mathcal{B}}_{D}$ are pairwise disjoint; $\Delta^{\mathcal{B}}_{\mathcal{O}}$ is a nonempty set of objects; $\Delta^{\mathcal{B}}_{\mathcal{R}}$ is a nonempty set of relationships, $\Delta^{\mathcal{B}}_{D} = \bigcup_{D_i \in \mathcal{D}} \Delta^{\mathcal{B}}_{D_i}$ is the set of values for all domains used in the schema Σ; and $\cdot^{\mathcal{B}}$ is a function that maps:

- Every domain symbol D_i to a set $D_i^{\mathcal{B}} = \Delta^{\mathcal{B}}_{D_i}$.
- Every object type symbol O, to a set $O^{\mathcal{B}} \subseteq \Delta^{\mathcal{B}}_{\mathcal{O}}$.
- Every relationship type symbol R to a set $R^{\mathcal{B}}$ of couples $\langle r, u \rangle$ where $r \in \Delta^{\mathcal{B}}_{\mathcal{R}_l}$ and u is a $\mathcal{U}$-labeled tuple over $\Delta^{\mathcal{B}}_{\mathcal{O}}$ such that if $\text{REL}(R) = \langle U_1 : O_1, \ldots, U_k : O_k \rangle$,

then: $\langle r, u\rangle \in R^{\mathcal{B}} \wedge u = \langle U_1 : o_1, \ldots, U_k : o_k\rangle \Rightarrow \forall i \in \{1, \ldots, k\}\ (o_i \in O_i^{\mathcal{B}})$. Further, $R^{\mathcal{B}}$ is such that: $\forall\, \langle r_1, u_1\rangle, \langle r_2, u_2\rangle \in R^{\mathcal{B}}\ (r_1 = r_2 \Rightarrow u_1 = u_2)$.

- Every attribute symbol A to a set $A^{\mathcal{B}} \subseteq (\Delta_{\mathcal{O}}^{\mathcal{B}} \cup \Delta_{\mathcal{R}}^{\mathcal{B}}) \times \Delta_{D}^{\mathcal{B}}$, such that, for each object or relationship type $X \in (\mathcal{O} \cup \mathcal{R})$, if $\text{ATT}(X)[A] = D_i$, then: $x \in X^{\mathcal{B}} \Rightarrow (\exists a_i \in D_i^{\mathcal{B}}\ (\langle x, a_i\rangle \in A^{\mathcal{B}}) \wedge \forall a_i\ (\langle x, a_i\rangle \in A^{\mathcal{B}} \Rightarrow a_i \in \Delta_{D_i}^{\mathcal{B}}))$. $\diamond$

Definition 3 (Consistent database state of a SimpleMads schema without perceptions)
A database state $\mathcal{B}$ is said to be consistent if it satisfies all of the constraints expressed in the schema:

- *Population inclusion*:
  $\forall X_1, X_2 \in (\mathcal{O} \cup \mathcal{R})\ (\text{ISA}(X_1, X_2) \Rightarrow X_1^{\mathcal{B}} \subseteq X_2^{\mathcal{B}})$.
- *Population intersection*:
  $\forall X_1, X_2 \in (\mathcal{O} \cup \mathcal{R})\ (X_1^{\mathcal{B}} \cap X_2^{\mathcal{B}} \neq \emptyset \Rightarrow X_1 = X_2 \vee \text{ISA}(X_1, X_2) \vee \text{ISA}(X_2, X_1) \vee$
  $\text{OVLP}\ (X_1, X_2))$
- *Cardinality constraints*:
  For each cardinality constraint $\text{CARD}(O, R, U)$ of a relationship $R \in \mathcal{R}$:

$$\forall o \in O^{\mathcal{B}}\ (\text{CMIN}(O, R, U) \leq \#\{\langle r, u\rangle \in R^{\mathcal{B}} \mid u[U] = o\} \leq \text{CMAX}(O, R, U)).$$

- *Key constraints*:
  For each key constraint $\text{KEY}(X, K)$ of an object or relationship type $X \in (\mathcal{O} \cup \mathcal{R})$, where $K = \{A_1, \ldots, A_n\}$:
  $\forall x_1, x_2 \in X^{\mathcal{B}}\ \forall i \in \{1, \ldots, n\}\ ((\langle x_1, a_i^1\rangle \in A_i^{\mathcal{B}} \wedge \langle x_2, a_i^2\rangle \in A_i^{\mathcal{B}} \wedge a_i^1 = a_i^2) \Rightarrow$
  $x_1 = x_2)$. $\diamond$

Proposition 1 (Logical implication for a SimpleMads schema without perceptions)
As a consequence of the definitions of SimpleMads schema and consistent database state, the following rule can be derived:

- *Inferred overlap links from is-a links*:
  $\forall X_1, X_2, X_3 \in (\mathcal{O} \cup \mathcal{R})\ (\text{ISA}(X_1, X_2) \wedge \text{ISA}(X_1, X_3) \Rightarrow \text{OVLP}(X_1, X_3))$.

5.2.3 Spatio-Temporal Modeling

In Mads, space and time description is orthogonal to data structure description, which means that the description of a phenomenon may be enhanced by spatial and temporal features whatever data structure (i.e., object, relationship, attribute) has been chosen to represent it. Mads allows describing spatial and temporal features with either a discrete or a continuous view. These are described next.

The *discrete view* (or *object view*) of space and time defines the spatial and temporal extents of the phenomena of interest. The spatial extent is the set of 2-dimensional or 3-dimensional points (defined by their geographical coordinates $\langle x, y\rangle$ or $\langle x, y, z\rangle$) that the phenomenon occupies in space. The temporal extent is the set of instants that the phenomenon occupies in time. Temporality in Mads corresponds to valid time, which conveys information on when a given fact, stored in the database, is considered valid from the application point of view.

Specific data types support the definition, manipulation, and querying of spatial and temporal values. Mads supports two hierarchies of dedicated data types, one for spatial data types, and one for temporal data types. Generic spatial (respectively, temporal) data types allow describing object types whose instances may have different types of spatial extents. For example, a River object type may contain instances for large rivers with an extent of type Surface and instances for small rivers with an extent of type Line. The Mads hierarchy of spatial data types is simpler that – while compatible with – the one proposed by the Open Geospatial Consortium [9]. Examples of spatial data types are: Geo (🌍), the most generic spatial data type, Surface (◆), and SurfaceBag (◆). The latter is useful for describing objects with a non-connected surface, like an archipelago. Examples of temporal data types are: Instant (○), Interval (◔), nand IntervalBag (⊛). The latter is useful for describing the periods of activity of non-continuous phenomena.

A spatial (temporal) object type is an object type that holds spatial (temporal) information pertaining to the object itself. For example, District is a spatial object type as shown by the surface (◆) icon on the right of its name, and LandPlot is a spatial and temporal object type with a lifespan of kind Interval (◔ icon on the left of its name). Following common practice, we call spatio-temporal an object type that either has both a spatial and a temporal extent, separately, or has a time-varying spatial extent, i.e., its spatial extent changes over time and the history of extent values is recorded (e.g., LandPlot). Similarly, spatial, temporal, and spatio-temporal relationship types hold spatial and/or temporal information pertaining to the relationship as a whole, exactly as for an object type. Time-varying and space-varying attributes are described hereinafter.

The spatial and temporal extents of an object (or relationship) type are kept in dedicated system-defined attributes: geometry for the spatial extent and lifecycle for the temporal extent. The attribute geometry is a spatial attribute (see below) with any spatial data type as domain. When representing a moving or deforming object (e.g., LandPlot), geometry is a time-varying spatial attribute. On the other hand, the attribute lifecycle allows database users to record when, in the real world, the object (or link) was (or is planned to be) created and deleted. It may also support recording that an object is temporarily suspended, like an employee who is on temporary leave. Therefore, the lifecycle of an instance says at each instant what is the status of the corresponding real-world object (or link): scheduled, active, suspended, or disabled.

A spatial (temporal) attribute is a simple attribute whose domain of values belongs to one of the spatial (temporal) data types. Each object and relationship type, whether spatial, temporal, or plain, may have spatial, temporal, and spatio-temporal attributes. For example, the BuiltUpLandPlot object type includes, in addition to its spatial extent (inherited from LandPlot), a complex and multivalued attribute buildings whose second component attribute, location, is a spatial attribute describing, for each building, its spatial extent, a surface. Practically, the implementation of a spatial attribute, as well as the one of a geometry attribute, varies according to the domain of the attribute. For instance, in 2D space a geometry of kind Point is usually implemented by a couple of coordinates $\langle x, y\rangle$ for each value, and a geometry of kind Surface by a list of couples $\langle x, y\rangle$ per value.

Spatial and temporal values for an object may have to be consistent with the spatial and temporal values of other related objects. *Constraining relationships* are binary relationships linking spatial (or temporal) object types stating that the geometries (or lifecycles) of the linked objects must comply with a spatial (or temporal) constraint. For example, Composes is both an aggregation and a constraining relationship of kind topological inclusion, as shown by the icon. The constraint states that a district and a land plot may be linked only if the spatial extent of the district effectively contains the spatial extent of the land plot. Produces is a synchronization relationship type of kind within (icon): It enforces the temporal extent of the Wine instance – an instant with year granularity describing the year of the wine – to be included within the temporal extent of the Vineyard instance – a time interval describing when the vineyard was productive.

Beyond the discrete view, there is a need to support another perception of space and time, the *continuous view* (or *field view*). In the continuous view a phenomenon is perceived as a function associating to each point (or instant) of a spatial (or temporal) extent a value. Mads supports the continuous view using space- and time-varying attributes, which are attributes whose value is a function that records the history – and possibly the future – of the value. The domain of the function is a spatial (and/or temporal) extent. Its range can be a set of simple values (e.g., Real for temperature, Point for a moving car), a set of composite values if the attribute is complex, and/or a powerset of values if the attribute is multivalued.

The object type District shows three examples of varying attributes and their visual notation in Mads (e.g., f(())). Attribute elevation is a space-varying attribute defined over the geometry of the district: It provides for each geographic point of the district its elevation. Attribute population is a time-varying attribute defined over a constant time interval, e.g., [1900-2007]. Attribute weather is a space and time-varying complex attribute which records for each point of the spatial extent of the district and for each instant of a constant time interval a composite value describing the weather at this location and this instant. Such space- and time-varying attributes are also called spatio-temporal attributes. As we have seen, the geometry attribute can also be time varying, like any spatial attribute. For instance, LandPlot has a time-varying geometry: any change of the spatial extent of land plots can therefore be recorded. Practically, the implementation of a continuous time-varying attribute is usually made up of (1) a list of ⟨instant, value⟩ pairs that records measured values (called sample values), and 2) a method that performs linear interpolation between two sample values to infer non-measured values. For instance, a time-varying point would be implemented by a list of triples $\langle \text{instant}, x, y\rangle$. On the other hand, time-varying attributes that are not continuous but that vary in a stepwise manner, like the geometry of LandPlot, are recorded by a list of couples ⟨time interval, value⟩.

A constraining topological relationship may link moving or deforming objects, i.e., spatial objects whose geometries are time-varying. An example is the topological inclusion relationship Composes that links District (a surface) and LandPlot (a time-varying surface). In this case two possible interpretations can be given to the topological predicate, depending on whether it must be satisfied either for at least one instant or for every instant belonging to the time extent of the varying geometries. Applied to the example of Fig. 5.1, this means that the relationship Composes can only link a District and a LandPlot instances such that their geometries intersect for at least one instant or for every instant of the temporal extent of the varying geometry of the land plot. When defining the relationship type, the designer has to specify which interpretation holds.

5.3 Perceptions

As explained in the introductory section of this chapter, the notion of perception in Mads captures a specific perspective that guides the definition of the corresponding content of the database. We first discuss the perception mechanism informally, and provide a formal definition afterwards. As Mads is intended for conceptual modeling, the definition of perceptions is dealt with as part of the conceptual design phase. The resulting conceptual schema will eventually be translated into logical and physical schemas. Perceptions, alike spatial and temporal features, will have to be implemented using the mechanisms provided by the target DBMS. We show in Sect. 5.8 a possible implementation of perceptions into the relational model.

Supporting multiple perceptions within the same database, as Mads does, means that different contents coexist in the database and the system knows how to identify and extract the content that corresponds to a specific perception (which we call *simple* perception) or to a combination of perceptions (which we call *composite* perception). For instance, the schema diagram in Fig. 5.3 illustrates a multiperception schema, separately showing the content of each of three simple perceptions designed to support information requirements from the wine makers, the wine experts, and the geologists working in the wine area. These perceptions are denoted $\mathsf{P_m}$, $\mathsf{P_e}$, and $\mathsf{P_g}$, respectively. The diagram uses visual duplication to show that the object type Wine belongs to two simple perceptions. Before accessing the database, a typical user transaction will specify which perception it wants to use, and will accordingly see the corresponding subset of the database.

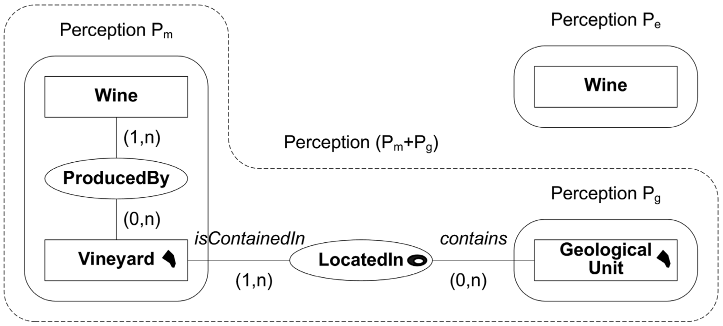


Fig. 5.3. A schema diagram showing three simple perceptions and one composite perception

Some applications may need to work simultaneously with data that has been defined as belonging to different perceptions. For example in Fig. 5.3, an application may wish to relate the geological information to the vineyard information, thus spanning over the geologist and wine makers perceptions. Such an application may wish to record the relationships between geological units in $\mathsf{P_g}$ and vineyards in $\mathsf{P_m}$, creating instances of the LocatedIn relationship type. Similarly, applications may need to simultaneously use different *representations* of the same phenomena belonging to different perceptions. For instance, in cartographic databases storing data for a set of maps representing the same region at different scales, it is common to organize the database as holding one simple perception per targeted scale. Yet there are applications that can compare the various representations in order to check their spatial consistency.

In summary, the perception mechanism must be able to support users using a single simple perception as well as users using data from multiple perceptions. It must also be able to support storing data that belong to a single perception, data that belong to multiple perceptions, and data that relate together data from different perceptions. We describe hereinafter a mechanism to respond to these requirements based on the combined use of simple and composite perceptions.

A *simple perception* provides an application with a view of the multiperception database that includes whatever data is defined as belonging to this perception, and nothing else. $\mathsf{P_m}$, $\mathsf{P_e}$, and $\mathsf{P_g}$, in Fig. 5.3 are simple perceptions. A multiperception database holds data belonging to various simple perceptions, say $(\mathsf{p_1}+\mathsf{p_2}+\ldots+\mathsf{p_n})$. A simple perception can be seen as a component of a multiperception database characterized by its own schema and its own instances (both materialized), respectively a subset of the multidatabase schema and instances. This subset is equivalent to a traditional database without the perception dimension. The various perceptions may differ in their scope, i.e., they may describe different sets of real-world entities and links, but these sets may also overlap and in databases they often do overlap in a large proportion. For instance, in Fig. 5.3 both perceptions $\mathsf{P_m}$ and $\mathsf{P_e}$ describe wines, possibly in different ways. The definition of simple perceptions is part of the schema design process, now ending up with the (basically static) definition of a multiperception schema.

Composite perceptions support working with data from multiple simple perceptions. A *composite perception* is dynamically defined by users depending on the information needs of their transactions. Transactions use an openDatabase command to specify which database they want to work with and which perception(s) they want to work with:

```
openDatabase(dbName, myView)
```

where myView is either a simple perception $\mathsf{p_i}$ or a composite perception denoted $(\mathsf{p_1}+\mathsf{p_2}+\ldots+\mathsf{p_k})$,

The view provided by a composite perception is created by the system on the fly and contains all the elements (schema and instances) of the component simple perceptions $\mathsf{p_i}$, plus the *interperception links* (relationships, is-a and overlap links, at the schema and instance levels), if any, that relate objects in different perceptions within

p_1, p_2, ..., and p_k (e.g., an object of p_1 and an object of p_2). For instance, in the Wine database of Fig. 5.3 perceptions P_m and P_g describe two disjoint parts of the real world, yet they are linked by a relationship type, LocatedIn. This relationship type, contrarily to ProducedBy, does not belong to any of the simple perceptions P_m, P_e, and P_g. It belongs only to the composite perception $(P_m + P_g)$. Therefore, while users of P_m see Wine, Vineyard, and ProducedBy, and users of P_g see GeologicalUnit, users of (P_m+P_g) see Wine, Vineyard, ProducedBy, GeologicalUnit, and LocatedIn.

Interperception links provide an explicit means to navigate between perceptions. By definition, they do not belong to any simple perception. For simplification purposes, we keep with the idea that every element belongs to at least a simple perception by considering that interperception links belong to a special simple perception that is system defined and not visible to users, and is denoted P_{ip}. Referring to Fig. 5.3, the ProducedBy relationship type links two P_m object types and is defined by the administrator as belonging to P_m: We say it is a *local* link. LocatedIn, instead, links a P_m object type and a P_g object type, and is therefore automatically identified as an interperception link, implicitly tagged P_{ip}. The set of local links and interperception links are disjoint. In the schema illustrated in Fig. 5.5, there is one relationship type ProducedBy which is local, even if it belongs to two perceptions, P_m and P_e. ProducedBy in the P_m (resp. P_e) perception links Wine objects that belong to P_m (resp. P_e) to Vineyard objects that also belong to P_m (resp. P_e). Should the application need to link, say, wines of P_m to vineyards of P_e, then the database administrators would have to define another relationship type, say WmProducedByVe, linking Wine objects of P_m to Vineyard objects of P_e.

The first step towards the creation of a multiperception database is for the database administrator to identify the set of simple perceptions that need to be explicitly defined (i.e., the set $SP = \{p_1, p_2, \ldots, p_n\}$). The following step for the database administrator is to define which data belongs to which perception. Any kind of schema element may have several representations. The population of an object or relationship type may also vary with the perception. Whatever methodology is used to perform this step (definitions organized by perception or by schema element), the result shall conform to the following:

- Each object and relationship type definition includes the specification of the perceptions it belongs to, and for each of these perceptions its corresponding representation, i.e., its attributes, and roles definitions.
- Each *interperception* relationship type, meant to connect objects in different perceptions, is defined as belonging to the peculiar perception P_{ip}.
- An element that has a single representation may belong to multiple perceptions. All its perceptions share its single representation. Conversely, an element can have multiple representations only if it belongs to at least as many perceptions (otherwise stated, an element has only one representations for each perception).
- Each perception has to denote a consistent database obeying the classical consistency rules for databases (e.g., no pending role in a relationship).

- To enforce perception consistency, an element a that is a component of an element b (e.g., an attribute of an object type or a component attribute of a complex attribute) can only belong to perceptions to which the b element belongs, as illustrated in the example of Fig. 5.4.

Figure 5.4 illustrates the definition of a multiperception object type, providing details about its perceptions, and attributes and keys for each of the two perceptions. The drawing of the two perceptions of the Wine object type as a single object type in which the two perceptions are merged is different from the drawing of the same Wine object type in Fig. 5.3 as two boxes, one per perception. Yet the difference only conveys the use of different visualization techniques. The information content is the same. Figure 5.4 directly corresponds to how Wine is defined using the Mads data definition language, which includes the definition of perceptions as part of the definition of each metadata element (i.e., data description element of the schema).

Figure 5.4 describes the representations of the Wine object type for the wine expert's perception P_e and for the wine maker's perception P_m. Attributes name, year, and wineType are common to both perceptions, with a common representation. Attributes degree and barrels are common to both perceptions, but they have a different definition (representation) for each perception and therefore their values will be different too. The value of degree is simplified (integer rather than real) for the perception P_e. Similarly, the attribute barrels is a simple Boolean attribute in perception P_e, stating if the wine has been kept in wooden barrels or not, while in perception P_m it is a complex attribute describing the time period during which the wine has been kept in barrels and the kind of wood of the barrels. Perception P_e has several attributes, rating, color, body, sugar, and food (the food matching the wine) that are specific to it and do not exist in P_m.

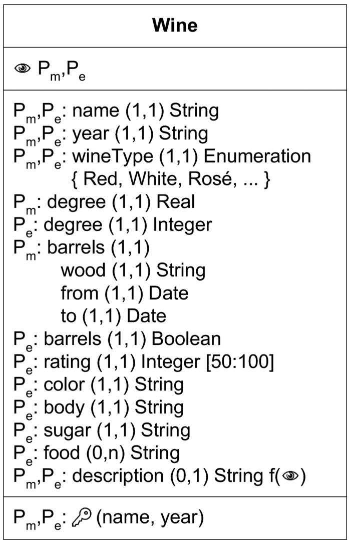

Wine
P_m,P_e
P_m,P_e: name (1,1) String
P_m,P_e: year (1,1) String
P_m,P_e: wineType (1,1) Enumeration { Red, White, Rosé, ... }
P_m: degree (1,1) Real
P_e: degree (1,1) Integer
P_m: barrels (1,1)
wood (1,1) String
from (1,1) Date
to (1,1) Date
P_e: barrels (1,1) Boolean
P_e: rating (1,1) Integer [50:100]
P_e: color (1,1) String
P_e: body (1,1) String
P_e: sugar (1,1) String
P_e: food (0,n) String
P_m,P_e: description (0,1) String f()
P_m,P_e: (name, year)

Fig. 5.4. The two perceptions of the Wine object type of Fig. 5.3

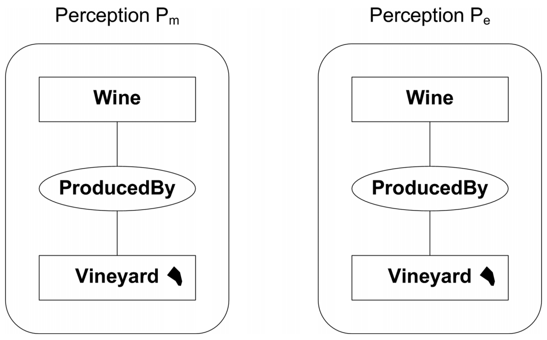


Fig. 5.5. A relationship type belonging to two perceptions

As shown, the representations hold by an object (relationship) type may have different sets of attributes, different characteristics for a common attribute (different cardinalities or value domains). Perceptions are also defined at the instance level. Therefore, an object (relationship) type belonging to several perceptions may have different sets of instances according to the perception. Similarly each instance (object or relationship) that belongs to several perceptions may have different values according to the perception. This is obvious when the sets of attributes are different for the various perceptions, but it is also true for an attribute with a unique definition. In this case, the value of the attribute depends upon the perception, and the attribute is said to be *perception-varying*. For example, in Fig. 5.4 the attribute description, recording a text of a few lines describing the wine, is perception-varying, as identified by the f(👁) icon. It has a unique definition common to both perceptions, but it has a different value for each perception, i.e., the text used by the wine maker is different from the text used by the wine expert.

Similarly to object types, when defining a relationship type the designer has to specify to which perceptions it belongs and for each perception its representation, i.e., defining its attributes and roles. Figure 5.5 illustrates two perceptions, P_m and P_e, sharing the relationship type ProducedBy and its linked object types Wine and Vineyard (this schema is different from the one illustrated Fig. 5.3). Let us assume that ProducedBy in P_m has a unique attribute quantity, and in P_e no attribute at all. Attribute quantity says how many kilograms of grapes harvested in this vineyard have been used for producing this wine. We also assume that the populations are different. In P_m, ProducedBy takes into account all contributing vineyards even if the quantity of grapes is small. In P_e, ProducedBy takes into account only the vineyards that have contributed to at least 15% of the total quantity of grapes used for producing this wine. Therefore, there is a constraint linking the two populations of ProducedBy as follows:

$$\text{population}(P_e.\text{ProducedBy}) \subseteq \text{population}(P_m.\text{ProducedBy})$$

that should be defined in the schema by an interperception is-a link.

Relationship types may have different representations for their roles (e.g., have different sets of roles according to the perception) and their semantics (e.g., being an

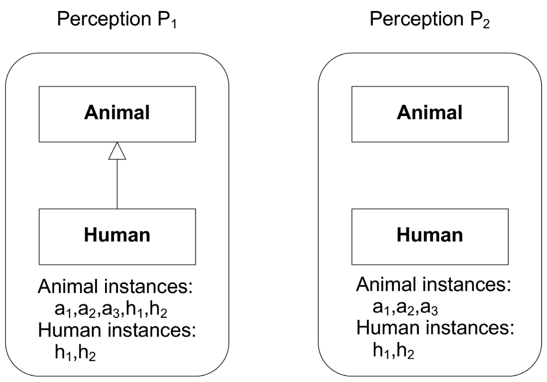


Fig. 5.6. Two perceptions that differ at the schema and the instance level (instances are symbolized by their oid)

aggregation relationship for a perception and a topological inclusion relationship for another).

Given two object types, their representations in different perceptions may be differently related, i.e., in a perception they may be related by an is-a link, in another perception they may be defined as disjoint, and yet in another one they may possibly overlap. They may also be linked by a relationship type in a perception, and not linked in another perception. Such flexibility is needed to allow independence between the perceptions.

Different representations for the same real-world entities and links may even contradict each other. For example, Fig. 5.6 shows a Mads schema with two perceptions. Perception P_1 considers humans to be a specific kind of animals. It therefore defines two object types, Human and Animal, linked by an is-a link making Human a subclass of Animal. Perception P_2 considers humans to be different from animals. It therefore contains another representation of the same two object types, possibly with different attributes and methods, where the two object types are by definition disjoint (they are not interrelated by a multi-instantiation link).

In terms of constraints, the database administrator can define interperception constraints on the value of attributes and on instances. Examples of usual constraints for an object or relationship type are: The set of instances – more precisely, the set of oids or rids – is the same for all perceptions, or, on the contrary, they are disjoint. Another constraint could state that the set of instances for a given perception is included in the set for another perception. An example could be in Fig. 5.6: "Every instance of Animal that has a representation in P_2 has also a representation in P_1." This should be defined in the schema by an interperception is-a.

Particularly important constraints are the identification constraints. There is indeed a need for being able to correlate and coordinate the various perceptions if required by application rules. For example, the cartographic application we already mentioned needs to be able to find all representations of an object (e.g., a building) to check their consistency (e.g., the point representing the spatial extent of the building in one perception $\mathsf{P_x}$ has to be inside the area representing the same building in another perception $\mathsf{P_y}$). Knowledge about the two representations of buildings is granted by the use of a composite perception $(\mathsf{P_x} + \mathsf{P_y})$, but this would not help if the user transaction is not able to identify, at the instance level, which $\mathsf{P_x}$ building is the same as a given $\mathsf{P_y}$ building. In our approach, the correlation between multiple representations of the same object (or relationship) relies on shared object (or relationship) identity, as is the case in semantic databases for the implementation of is-a links. All representations of an (object or relationship) instance share the same oid (or rid in case of a relationship instance), which is defined by the system. Identity provides the shared property that links together all the representations of the same instance. As in object-oriented systems, relying on identity, rather than on user-defined keys, guarantees that the system can keep a correct understanding of instances even if users enter erroneous data in the database.

Identity, however, is not enough. How would the system know that the $\mathsf{P_x}$ user inserting a building new to her is actually creating her representation of a building already inserted in the database by a $\mathsf{P_y}$ user? One solution would be to enforce that instances of shared object types, e.g., Building in both $\mathsf{P_x}$ and $\mathsf{P_y}$, can only be created by users with the composite perception $(\mathsf{P_x} + \mathsf{P_y})$. This solution is in our opinion overly restrictive. We prefer the solution (adopted by relational DBMS) where users of multiperception elements rely on a shared identification mechanism, i.e., a shared key, to correlate the multiple representations of the same object or relationship. This solution is presented in more detail in Sect. 5.7.

5.4 More on Interperceptions Links

As we have already seen, two simple perceptions may describe either the same part of the real world, or disjoint or overlapping parts. The common part may be described by different representations of the same object types, like the two representations of Wine for perceptions $\mathsf{P_m}$ and $\mathsf{P_e}$ in Figs. 5.3 and 5.4, or the two representations of Animal for perceptions $\mathsf{P_1}$ and $\mathsf{P_2}$ in Fig. 5.6. However, a set of real-world entities may also be described in different perceptions by different object types. For example, Fig. 5.7 shows a variant of Fig. 5.3 where the wine expert's perception $\mathsf{P_e}$, instead of providing a generic Wine object type provides three disjoint object types RedWine, WhiteWine, and RoseWine.

In this kind of situation where some instances of two object types belonging to two different perceptions describe the same real-world entities, designers using the Mads data model have two possibilities:

- If the mapping between the instances of the two object types is injective on both sides, the object types may be defined as sharing oids, i.e., an *interperception is-a* or *overlap link* can relate the two object types. As shown in Fig. 5.7, if we assume that all wines described in $\mathsf{P_e}$ are also described in $\mathsf{P_m}$ in the Wine object type, designers may assert:

  $\mathsf{P_e}$.WhiteWine is-a $\mathsf{P_m}$.Wine
  $\mathsf{P_e}$.RedWine is-a $\mathsf{P_m}$.Wine
  $\mathsf{P_e}$.RoseWine is-a $\mathsf{P_m}$.Wine

- If the mapping between the instances is not injective, i.e., an instance of an object type may correspond to several instances of the other object type, designers may relate these object types through an interperception relationship type. Mads supports a specific kind of semantics for these relationship types that link objects representing the same real-world entities, the *inter-representation* semantics. For example, let us assume a perception containing an object type Person and another one containing an object type Marriage. Designers could relate these two object types through an interperception and inter-representation relationship type that would link each instance of Marriage to two instances of Person, the husband and the wife.

Notice that, in the case of a mapping that is injective on both sides, designers may choose between the two solutions: either an interperception multi-instantiation link (is-a or overlap according to the cardinalities of the mapping) or an interperception and inter-representation relationship type. If the cardinalities of the mapping are (1,1)–(0,1), the is-a link is equivalent to an inter-representation relationship type with the same cardinalities. However, the is-a link is a more direct representation of the semantics of the mapping and hence should be preferred. If the mapping is (0,1)–(0,1), the two solutions, an interperception overlap link and an interperception and inter-representation relationship type, are not equivalent. In the latter, each simple perception may create instances in its object type without worrying about the other perception. Afterwards, users of the composite perception can create the interperception and inter-representation relationship instances that will link together the corresponding instances of the two object types. On the other hand in the former solution, the interperception overlap, the creation of an instance i_1 in one object type, requires to know if the corresponding instance, say i_2, already exists in the other perception because the insertion of i_1 requires the oid of i_2, in order to create i_1 with the same oid.

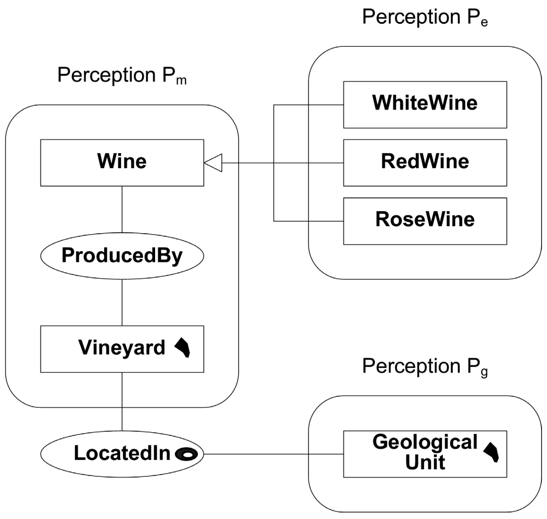


Fig. 5.7. Interperception is-a links

In summary, Mads supports both interperception multi-instantiation links and interperception relationship types, thus allowing designers to explicitly describe many kinds of situations where the real world described by two perceptions overlap.

5.5 Formal Definition of a SimpleMads Multiperception Database

In this section we give a formal definition of a Mads multiperception database. If all specifications related to perceptions are taken out of the following definition, the definition reduces to the definition of a Mads database without perceptions, which we provided in Sect. 5.2.2. We keep here the same simplifying assumptions as in Sect. 5.2.2. For sake of simplicity, we omit in this formalization the definition of perception-varying attributes.

The model-theoretic semantics associated with the SimpleMads model with the perception dimension is given next.

Definition 4 (SimpleMads multiperception schema)
A SimpleMads multiperception schema is a tuple: $\Sigma = (\mathcal{L}, \text{PERC}, \text{REL}_l, \text{REL}_{ip}, \text{ATT}, \text{CARD}_l, \text{CARD}_{ip}, \text{ISA}_l, \text{ISA}_{ip}, \text{OVLP}_l, \text{OVLP}_{ip}, \text{KEY})$, such that:

- $\mathcal{L}$ is a finite alphabet partitioned into the sets: $\mathcal{P}$ (*perception* symbols), $\mathcal{O}$ (*object type* symbols), $\mathcal{R}$ (*relationship type* symbols), $\mathcal{A}$ (*attribute* symbols), $\mathcal{U}$ (*role* symbols), and $\mathcal{D}$ (*domain* symbols). Further, $\mathcal{R}$ is partitioned into the sets $\mathcal{R}_l$ and $\mathcal{R}_{ip}$ denoting, respectively, the *local* and the *interperception* relationship type symbols. Also, $\mathcal{P} = \{P_{ip}\} \cup \mathcal{P}_s$ where P_{ip} is a peculiar perception to which are attached all interperception relationship types and $\mathcal{P}_s$ is the set of simple perception symbols.
- PERC (perceptions) is a total function that maps each object or relationship type symbol X in $\mathcal{O} \cup \mathcal{R}$ to a nonempty set of perceptions $\text{PERC}(X) \subseteq 2^{\mathcal{P}}$ such that $\forall X \in (\mathcal{O} \cup \mathcal{R}_l)\ (\text{PERC}(X) \subseteq \mathcal{P}_s \wedge \text{PERC}(X) \neq \emptyset)$ and $\forall R \in \mathcal{R}_{ip}\ (\text{PERC}(R) = \{P_{ip}\})$.
- REL_l (local relationships) is a total function that maps a couple made up of a local relationship type symbol R in $\mathcal{R}_l$ and a perception symbol P in $\text{PERC}(R)$ to an $\mathcal{U}$-labeled tuple over $\mathcal{O}$, $\text{REL}_l(R,P) = \langle U_1 : O_1, \ldots, U_k : O_k \rangle$, where $k \geq 2$ is the *arity* of R in P, and $\forall i \in \{1, \ldots, k\}\ (P \in \text{PERC}(O_i))$.
- REL_{ip} (interperception relationships) is a total function that maps an interperception relationship type symbol R in $\mathcal{R}_{ip}$ to an $\mathcal{U} \times \mathcal{P}_s$-labeled tuple over $\mathcal{O}$, $\text{REL}_{ip}(R) = \langle (U_1, P_1) : O_1, \ldots, (U_k, P_k) : O_k \rangle$, where $k \geq 2$ is the *arity* of R, and $\forall i \in \{1, \ldots, k\}\ (P_i \in \text{PERC}(O_i))$.
- ATT (attributes) is a partial function that maps a couple made up of an object or relationship type symbol X in $\mathcal{O} \cup \mathcal{R}$ and a perception symbol P in $\text{PERC}(X)$ to an $\mathcal{A}$-labeled tuple over $\mathcal{D}$, $\text{ATT}(X,P) = \langle A_1 : D_1, \ldots, A_h : D_h \rangle$.
- CARD_l (local cardinalities) is a partial function $\mathcal{O} \times \mathcal{R}_l \times \mathcal{U} \times \mathcal{P}_s \rightarrow \mathbb{N} \times (\mathbb{N} \cup \{\infty\})$ that defines cardinality constraints associated to the roles of the local relationship types in perceptions. For a local relationship type R and one of its perceptions P such that $\text{REL}_l(R,P) = \langle U_1 : O_1, \ldots, U_k : O_k \rangle$, we use $\text{CMIN}_l(O_i, R, U_i, P)$ and $\text{CMAX}_l(O_i, R, U_i, P)$ to denote the first and second component of CARD_l.

- CARD_{ip} (interperception cardinalities) is a partial function $\mathcal{O} \times \mathcal{R}_{ip} \times \mathcal{U} \times \mathcal{P}_s \to \mathbb{N} \times (\mathbb{N} \cup \{\infty\})$ that defines cardinality constraints associated to the roles of interperception relationship types. For an interperception relationship type R such that $\text{REL}_{ip}(R) = \langle (U_1, P_1) : O_1, \ldots, (U_k, P_k) : O_k \rangle$, we use $\text{CMIN}_{ip}(O_i, R, U_i, P_i)$ and $\text{CMAX}_{ip}(O_i, R, U_i, P_i)$ to denote the first and second component of CARD_{ip}.
- ISA_l (local is-a links) is a ternary relation $\text{ISA}_l \subseteq (\mathcal{O} \times \mathcal{O} \times \mathcal{P}_s) \cup (\mathcal{R}_l \times \mathcal{R}_l \times \mathcal{P}_s)$ that defines is-a links for object and relationship types in each perception. The transitive closure ISA_l^+ of ISA_l in each perception is defined as follows

  $$\begin{aligned}&\forall X_1, X_2, X_3 \in (\mathcal{O} \cup \mathcal{R}_l)\ \forall P \in \mathcal{P}_s\\ &\quad \text{ISA}_l(X_1, X_2, P) \Rightarrow P \in \text{PERC}(X_1) \cap \text{PERC}(X_2)\\ &\quad \text{ISA}_l(X_1, X_2, P) \Rightarrow \text{ISA}_l^+(X_1, X_2, P)\\ &\quad \text{ISA}_l^+(X_1, X_2, P) \wedge \text{ISA}_l^+(X_2, X_3, P) \Rightarrow \text{ISA}_l^+(X_1, X_3, P).\end{aligned}$$

- ISA_{ip} (interperception is-a links) is a quaternary relation $\text{ISA}_{ip} \subseteq (\mathcal{O} \times \mathcal{P}_s \times \mathcal{O} \times \mathcal{P}_s) \cup (\mathcal{R}_l \times \mathcal{P}_s \times \mathcal{R}_l \times \mathcal{P}_s)$ that defines is-a links for object and relationship types belonging to different perceptions. ISA_{ip} is such that:

  $$\forall X_1, X_2 \in (\mathcal{O} \cup \mathcal{R}_l)\ \forall P_1, P_2 \in \mathcal{P}_s\ (\text{ISA}_{ip}(X_1, P_1, X_2, P_2) \Rightarrow P_1 \neq P_2).$$

  The transitive closure ISA_{ip}^+ of ISA_{ip} is defined as follows:

  $$\begin{aligned}&\forall X_1, X_2, X_3 \in (\mathcal{O} \cup \mathcal{R}_l)\ \forall P_1, P_2, P_3 \in \mathcal{P}_s\\ &\quad \text{ISA}_{ip}(X_1, P_1, X_2, P_2) \Rightarrow \text{ISA}_{ip}^+(X_1, P_1, X_2, P_2)\\ &\quad \text{ISA}_{ip}^+(X_1, P_1, X_2, P_2) \wedge \text{ISA}_{ip}^+(X_2, P_2, X_3, P_3) \wedge P_1 \neq P_3 \Rightarrow\\ &\text{ISA}_{ip}^+(X_1, P_1, X_3, P_3).\end{aligned}$$

- OVLP_l (local overlap links) is a ternary relation $\text{OVLP}_l \subseteq (\mathcal{O} \times \mathcal{O} \times \mathcal{P}_s) \cup (\mathcal{R}_l \times \mathcal{R}_l \times \mathcal{P}_s)$ that defines overlapping links between object or relationship types for each perception.

  $$\text{OVLP}_l(X_1, X_2, P) \Rightarrow P \in \text{PERC}(X_1) \cap \text{PERC}(X_2)$$

  OVLP_l is symmetric for each perception:

  $$\forall X_1, X_2 \in (\mathcal{O} \cup \mathcal{R}_l)\ \forall P \in \mathcal{P}_s\ (\text{OVLP}_l(X_1, X_2, P) \Rightarrow \text{OVLP}_l(X_2, X_1, P)).$$

- OVLP_{ip} (interperception overlap links) is a quaternary relation $\text{OVLP}_{ip} \subseteq (\mathcal{O} \times \mathcal{P}_s \times \mathcal{O} \times \mathcal{P}_s) \cup (\mathcal{R}_l \times \mathcal{P}_s \times \mathcal{R}_l \times \mathcal{P}_s)$ that defines overlapping links between object or relationship types belonging to different perceptions. OVLP_{ip} is such that:

  $$\forall X_1, X_2 \in (\mathcal{O} \cup \mathcal{R}_l)\ \forall P_1, P_2 \in \mathcal{P}_s\ (\text{OVLP}_{ip}(X_1, P_1, X_2, P_2) \Rightarrow P_1 \neq P_2).$$

  Further, OVLP_{ip} is symmetric:

  $$\begin{aligned}&\forall X_1, X_2 \in (\mathcal{O} \cup \mathcal{R}_l)\ \forall P_1, P_2 \in \mathcal{P}_s\ (\text{OVLP}_{ip}(X_1, P_1, X_2, P_2) \Rightarrow\\ &\quad \text{OVLP}_{ip}\ (X_2, P_2, X_1, P_1)).\end{aligned}$$

- KEY is a ternary relation, $\text{KEY} \subseteq (\mathcal{O} \cup \mathcal{R}) \times \mathcal{P} \times 2^{\mathcal{A}}$, which associates to each object and relationship type symbol and one of its given perception a set of keys, each key being composed of a set of attributes of the object or relationship type for the perception. $\diamond$

The model-theoretic semantics associated with the SimpleMads model with the perception dimension is given next.

Definition 5 (Database state of a SimpleMads multiperception schema)
Let Σ be a SimpleMads multiperception schema. A *database state* for the schema Σ is a tuple $\mathcal{B} = (\Delta^{\mathcal{B}}_{\mathcal{O}} \cup \Delta^{\mathcal{B}}_{\mathcal{R}} \cup \Delta^{\mathcal{B}}_{D}, \cdot^{\mathcal{B}(P)})$, such that: the three sets $\Delta^{\mathcal{B}}_{\mathcal{O}}$, $\Delta^{\mathcal{B}}_{\mathcal{R}}$, and $\Delta^{\mathcal{B}}_{D}$ are pairwise disjoint; $\Delta^{\mathcal{B}}_{\mathcal{O}}$ is a nonempty set of objects; $\Delta^{\mathcal{B}}_{\mathcal{R}} = \Delta^{\mathcal{B}}_{\mathcal{R}_l} \cup \Delta^{\mathcal{B}}_{\mathcal{R}_{ip}}$ is a nonempty set of local and interperception relationships, where $\Delta^{\mathcal{B}}_{\mathcal{R}_l}$ and $\Delta^{\mathcal{B}}_{\mathcal{R}_{ip}}$ are disjoint; $\Delta^{\mathcal{B}}_{D} = \bigcup_{D_i \in \mathcal{D}} \Delta^{\mathcal{B}}_{D_i}$ is the set of values for all domains used in the schema Σ; and $\cdot^{\mathcal{B}(P)}$ is a function that, for some $P \in \mathcal{P}$, maps:

- Every domain symbol D_i, for every simple perception $P \in \mathcal{P}_s$, into a set $D_i^{\mathcal{B}(P)} = \Delta^{\mathcal{B}}_{D_i}$, such that:

  $\forall P_1, P_2 \in \mathcal{P}_s \; (D_i^{\mathcal{B}(P_1)} = D_i^{\mathcal{B}(P_2)})$.
- Every object type symbol O, for any of its perceptions $P \in \text{PERC}(O)$, to a set $O^{\mathcal{B}(P)} \subseteq \Delta^{\mathcal{B}}_{\mathcal{O}}$.
- Every local relationship type symbol R, for any of its perceptions $P \in \text{PERC}(R)$, to a set $R^{\mathcal{B}(P)}$ of couples $\langle r, u \rangle$ where $r \in \Delta^{\mathcal{B}}_{\mathcal{R}_l}$ and u is a $\mathcal{U}$-labeled tuple over $\Delta^{\mathcal{B}}_{\mathcal{O}}$ such that if $\text{REL}_l(R, P) = \langle U_1 : O_1, \ldots, U_k : O_k \rangle$, then: $\langle r, u \rangle \in R^{\mathcal{B}(P)} \wedge u = \langle U_1 : o_1, \ldots, U_k : o_k \rangle \Rightarrow \forall i \in \{1, \ldots, k\} \; (o_i \in O_i^{\mathcal{B}(P)})$. Further, $R^{\mathcal{B}(P)}$ is such that:

  $\forall \langle r_1, u_1 \rangle, \langle r_2, u_2 \rangle \in R^{\mathcal{B}(P)} \; (r_1 = r_2 \Rightarrow u_1 = u_2)$.
- Every interperception relationship type symbol R, for the P_{ip} perception, to a set $R^{\mathcal{B}(P)}$ of couples $\langle r, u \rangle$ where $r \in \Delta^{\mathcal{B}}_{\mathcal{R}_{ip}}$ and u is a $\mathcal{U} \times \mathcal{P}_s$-labeled tuple over $\Delta^{\mathcal{B}}_{\mathcal{O}}$ such that if $\text{REL}_{ip}(R) = \langle (U_1, P_1) : O_1, \ldots, (U_k, P_k) : O_k \rangle$, then: $\langle r, u \rangle \in R^{\mathcal{B}(P)} \wedge u = \langle (U_1, P_1) : o_1, \ldots, (U_k, P_k) : o_k \rangle \Rightarrow \forall i \in \{1, \ldots, k\} \; (o_i \in O_i^{\mathcal{B}(P_i)})$. Further, $R^{\mathcal{B}(P)}$ is such that:

  $\forall \langle r_1, u_1 \rangle, \langle r_2, u_2 \rangle \in R^{\mathcal{B}(P)} \; (r_1 = r_2 \Rightarrow u_1 = u_2)$.
- Every attribute symbol A, for a perception $P \in \mathcal{P}$, to a set $A^{\mathcal{B}(P)} \subseteq (\Delta^{\mathcal{B}}_{\mathcal{O}} \cup \Delta^{\mathcal{B}}_{\mathcal{R}}) \times \Delta^{\mathcal{B}}_{D}$, such that, for each object or relationship type $X \in (\mathcal{O} \cup \mathcal{R})$ and perception $P \in \mathcal{P}$, if $\text{ATT}(X, P)[A] = D_i$, then: $x \in X^{\mathcal{B}(P)} \Rightarrow (\exists a_i \in D_i^{\mathcal{B}} \; (\langle x, a_i \rangle \in A^{\mathcal{B}(P)}) \wedge \forall a_i \; (\langle x, a_i \rangle \in A^{\mathcal{B}(P)} \Rightarrow a_i \in \Delta^{\mathcal{B}}_{D_i}))$. $\diamond$

Definition 6 (Consistent database state of a multiperception SimpleMads schema)
A database state $\mathcal{B}$ is said to be consistent if it satisfies all of the constraints expressed in the schema:

- *Local population inclusion*:

  $\forall X_1, X_2 \in (\mathcal{O} \cup \mathcal{R}_l) \; \forall P \in \mathcal{P}_s \; (\text{ISA}_l(X_1, X_2, P) \Rightarrow X_1^{\mathcal{B}(P)} \subseteq X_2^{\mathcal{B}(P)})$.

- *Interperception population inclusion*:

  $\forall X_1, X_2 \in (\mathcal{O} \cup \mathcal{R}_l)\ \forall P_1, P_2 \in \mathcal{P}_s\ (\textsc{isa}_{ip}(X_1, P_1, X_2, P_2) \Rightarrow X_1^{\mathcal{B}(P_1)} \subseteq X_2^{\mathcal{B}(P_2)})$.

- *Local population intersection*:

  $\forall X_1, X_2 \in (\mathcal{O} \cup \mathcal{R}) \forall P \in \mathcal{P}_s\ (X_1^{\mathcal{B}(P)} \cap X_2^{\mathcal{B}(P)} \neq \emptyset \Rightarrow X_1 = X_2 \vee$
  $\textsc{isa}_l^+(X_1, X_2, P) \vee \textsc{isa}_l^+(X_2, X_1, P) \vee \textsc{ovlp}_l(X_1, X_2, P))$.

- *Interperception population intersection*:

  $\forall X_1, X_2 \in (\mathcal{O} \cup \mathcal{R})\ \forall P_1, P_2 \in \mathcal{P}_s\ (X_1^{\mathcal{B}(P_1)} \cap X_2^{\mathcal{B}(P_2)} \neq \emptyset \wedge P_1 \neq P_2 \Rightarrow X_1 = X_2 \vee$
  $\textsc{isa}_{ip}^+(X_1, P_1, X_2, P_2) \vee \textsc{isa}_{ip}^+(X_2, P_2, X_1, P_1) \vee \textsc{ovlp}_{ip}(X_1, P_1, X_2, P_2))$.

- *Local cardinality constraints*:
  For each cardinality constraint $\textsc{card}_l(O, R, U, P)$ of a local relationship $R \in \mathcal{R}_l$ and a perception $P \in \textsc{perc}(R)$: $\forall o \in O^{\mathcal{B}(P)}\ (\textsc{cmin}_l(O, R, U, P) \leq \#\{\langle r, u\rangle \in R^{\mathcal{B}(P)} \mid u[U] = o\} \leq \textsc{cmax}_l(O, R, U, P))$.
- *Interperception cardinality constraints*:
  For each cardinality constraint $\textsc{card}_{ip}(O, R, U, P)$ of an interperception relationship $R \in \mathcal{R}_{ip}$ and a perception $P \in \textsc{perc}(O)$: $\forall o \in O^{\mathcal{B}(P)}$
  $(\textsc{cmin}_{ip}(O, R, U, P) \leq \#\{\langle r, u\rangle \in R^{\mathcal{B}(P)} \mid u[(U, P)] = o\} \leq \textsc{cmax}_{ip}(O, R, U, P))$.
- *Key constraints*:
  For each key constraint $\textsc{key}(X, P, K)$ of an object or relationship type $X \in (\mathcal{O} \cup \mathcal{R})$ in a perception $P \in \mathcal{P}$, where $K = \{A_1, \ldots, A_n\}$: $x_1, x_2 \in X^{\mathcal{B}(P)} \Rightarrow (\forall i \in \{1, \ldots, n\}\ (\langle x_1, a_i^1\rangle \in A_i^{\mathcal{B}(P)} \wedge \langle x_2, a_i^2\rangle \in A_i^{\mathcal{B}(P)} \wedge a_i^1 = a_i^2) \Rightarrow x_1 = x_2)$.
- *Attributes common to several perceptions*:
  For each $X \in (\mathcal{O} \cup \mathcal{R}_l)$, for each $P_1, P_2 \in \textsc{perc}(X)$, for each $A \in \mathcal{A}$, for each $D \in \mathcal{D}$ such that $\textsc{att}(X, P_1)[A] = \textsc{att}(X, P_2)[A] = D$: $\forall x \in (X^{\mathcal{B}(P_1)} \cap X^{\mathcal{B}(P_2)})\ \forall \langle x, a_1\rangle \in A^{\mathcal{B}(P_1)}\ \forall \langle x, a_2\rangle \in A^{\mathcal{B}(P_2)}\ (a_1 = a_2)$.
- *Roles common to several perceptions*:
  For each $R \in \mathcal{R}_l$, for each $P_1, P_2 \in \textsc{perc}(R)$, for each $U \in \mathcal{U}$ such that $\textsc{rel}_l(R, P_1)[U] = \textsc{rel}_l(R, P_2)[U]$:

  $\forall \langle r_1, u_1\rangle \in R^{\mathcal{B}(P_1)}\ \forall \langle r_2, u_2\rangle \in R^{\mathcal{B}(P_2)}\ (r_1 = r_2 \Rightarrow u_1[U] = u_2[U])$. $\diamond$

5.6 Dependencies between Perceptions

In databases, the existence of some elements may depend upon other ones (let us call these other elements the reference elements): As databases follow the closed-world assumption, existence-dependent elements cannot be created if these reference elements do not exist, and conversely the deletion of a reference element has to be propagated to its dependent elements or prevented as long as it has dependants. This is the case of all relationship instances: A relationship instance cannot exist without the objects that it links. The other elements that are existence dependent are: object types linked to a relationship type (the reference element) through a mandatory role, and object and relationship types that have one or several super-types (the reference elements). Classic (i.e., without perception) database systems, which assume the closed-world assumption, enforce these existence constraints. When dealing with a multiple perceptions and representations database, if the dependent element, say DE, belongs to a perception, say P_1, while the reference element, say RE, belongs to another one, say P_2, then insertions of instances of DE cannot be local operations in P_1, and deletions of instances of RE cannot be local operations in P_2. We say that the perceptions P_1 and P_2 are mutually dependent. Insertions of instances of the dependent element and deletions of the reference element require using the composite perception $(P_1 + P_2)$.

For example in Fig. 5.7, the perceptions P_m and P_g are mutually dependent because the cardinalities of the relationship type LocatedIn (shown in Fig. 5.9) say that each Vineyard object must be linked to at least one GeologicalUnit object. This implies that, when creating a Vineyard object in P_m, it should be linked straight away to a GeologicalUnit object of P_g, thus requiring the composite perception $(P_m + P_g)$. On the other hand, a GeologicalUnit object of P_g can be deleted only if it is no longer linked to Vineyard objects of P_m, or the deletion should be propagated to the Vineyard objects, which requires the $(P_m + P_g)$ perception. The perceptions P_m and P_e are also mutually dependent for the creation of instances of WhiteWine, RedWine, and RoseWine and for the deletion of instances of Wine.

An interperception overlap link also creates a dependency between the perceptions, because – as we have seen in previously – adding a new instance to an already-existing object requires knowing its oid. For example, let us assume a variant of Fig. 5.7 where WhiteWine, RedWine, and RoseWine of perception P_e describe some wines of perception P_m but also some other wines not recorded in P_m. Let us assume that the designer expresses this knowledge by three interperception overlap links:

Overlap (P_m.Wine, P_e.WhiteWine)
Overlap (P_m.Wine, P_e.RedWine)
Overlap (P_m.Wine, P_e.RoseWine)

Then, an insertion of a wine in either perception, P_m or P_e, requires to access the other perception in order to know if the wine already exists and then get its oid. The two perceptions are mutually dependent for insertions. Yet deletions are local operations.

In conclusion, any interperception link, be it a relationship type, an is-a or an overlap link, causes a dependency between the two perceptions for the insertion or deletion of the linked elements.

Another kind of dependency between perceptions is the propagation of reasoning from one perception to another one. The reasoning that takes place in Mads schemas without perceptions (inferred overlap links from is-a links) is extended to multiperception Mads schemas. It allows to infer a local is-a link from two interperception is-a links as shown in Fig.5.8, exactly like in distributed description logics where generalized subsumptions are propagated between modules through bridge rules (see

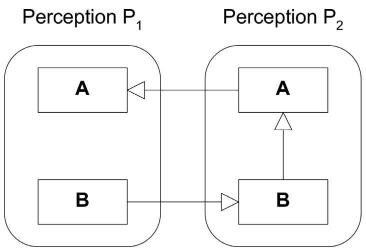


Fig. 5.8. Is-a propagation from one perception to another one

Chapter 12 in this book). In the same way, local and interperception overlap links are inferred from interperception is-a links.

Proposition 2 (Logical implication for a SimpleMads multiperception schema) As a consequence of the definitions of SimpleMads multiperception schema and consistent database state, the following rules can be derived:

- *Inferred local overlap links from local is-a links*:

  $\forall X_1, X_2, X_3 \in (\mathcal{O} \cup \mathcal{R}_l)\ \forall P \in \mathcal{P}_s$
  $(\text{ISA}_l(X_1, X_2, P) \wedge \text{ISA}_l(X_1, X_3, P) \Rightarrow \text{OVLP}_l(X_1, X_3, P)).$

- *Inferred local is-a links from interperception is-a links*:

  $\forall X_1, X_2, X_3 \in (\mathcal{O} \cup \mathcal{R}_l)\ \forall P_1, P_2 \in \mathcal{P}_s$
  $(\text{ISA}^+_{ip}(X_1, P_1, X_2, P_2) \wedge \text{ISA}^+_{ip}(X_2, P_2, X_3, P_1) \Rightarrow \text{ISA}_l(X_1, X_3, P_1)).$

- *Inferred local overlap links from interperception is-a links*:

  $\forall X_1, X_2, X_3 \in (\mathcal{O} \cup \mathcal{R}_l)\ \forall P_1, P_2 \in \mathcal{P}_s$
  $(\text{ISA}^+_{ip}(X_1, P_1, X_2, P_2) \wedge \text{ISA}^+_{ip}(X_1, P_1, X_3, P_2) \Rightarrow \text{OVLP}_l(X_2, X_3, P_2)).$

- *Inferred interperception overlap links from interperception is-a links*:

  $\forall X_1, X_2, X_3 \in (\mathcal{O} \cup \mathcal{R}_l)\ \forall P_1, P_2, P_3 \in \mathcal{P}_s$
  $(\text{ISA}^+_{ip}(X_1, P_1, X_2, P_2) \wedge \text{ISA}^+_{ip}(X_1, P_1, X_3, P_3) \Rightarrow \text{OVLP}_{ip}(X_2, P_2, X_3, P_3)).$

5.7 Using Perceptions

As already stated, user interaction with a multiperception database starts with the specification of which perception the user wants to work with. The following Mads command provides this functionality:

```
openDatabase(dbName, myView)
```

where dbName is the name of a multiperception database and myView denotes either a simple perception p_i or a composite perception $(\text{p}_1 + \text{p}_2 + \ldots + \text{p}_\text{k})$.

Upon receiving this command, the system creates a new virtual database (its schema and instantiation) out of the multiperception database dbName. This new virtual database is the "view" provided by the perception myView to the user. If myView contains a composite perception the virtual database will be a true multiperception database, otherwise it will be a monoperception database, equivalent to a classic database without perception. Hereinafter, we formally define the schema and semantics of a perception, be it a simple perception or a composite one.

Definition 7 (Schema and semantics of a perception)
Let $\Sigma = (\mathcal{L}, \textsc{perc}, \textsc{rel}_l, \textsc{rel}_{ip}, \textsc{att}, \textsc{card}_l, \textsc{card}_{ip}, \textsc{isa}_l, \textsc{isa}_{ip}, \textsc{ovlp}_l, \textsc{ovlp}_{ip}, \textsc{key})$ be a SimpleMads multiperception schema, where $\mathcal{L} = \mathcal{P} \cup \mathcal{O} \cup \mathcal{R} \cup \mathcal{A} \cup \mathcal{U} \cup \mathcal{D}$, $\mathcal{P} = \{P_{ip}\} \cup \mathcal{P}_s$, $\mathcal{R} = \mathcal{R}_l \cup \mathcal{R}_{ip}$. Let $\mathcal{B} = (\Delta^{\mathcal{B}}_{\mathcal{O}} \cup \Delta^{\mathcal{B}}_{\mathcal{R}} \cup \Delta^{\mathcal{B}}_{D}, \cdot^{\mathcal{B}(P)})$ be a consistent database state for Σ. Let $\mathcal{P}'_s$ be a nonempty set of perception symbols, $\mathcal{P}'_s \subset \mathcal{P}_s$. The perception $\mathcal{P}'_s$ of the multiperception database $(\Sigma, \mathcal{B})$ is a SimpleMads multiperception database, whose schema Σ' and database state $\mathcal{B}'$ are defined as follows:

$\Sigma' = (\mathcal{L}', \textsc{perc}', \textsc{rel}'_l, \textsc{rel}'_{ip}, \textsc{att}', \textsc{card}'_l, \textsc{card}'_{ip}, \textsc{isa}'_l, \textsc{isa}'_{ip}, \textsc{ovlp}'_l, \textsc{ovlp}'_{ip}, \textsc{key}')$, is defined by:

- $\mathcal{L}' \subset \mathcal{L}$ is the finite alphabet partitioned into the sets $\mathcal{P}'$, $\mathcal{O}'$, $\mathcal{R}'$, $\mathcal{A}'$, $\mathcal{U}'$, and $\mathcal{D}'$ defined by:

  $$\begin{aligned}
  &\mathcal{P}' = \{P_{ip}\} \cup \mathcal{P}'_s,\\
  &\mathcal{O}' = \{O \mid O \in \mathcal{O} \wedge \textsc{perc}(O) \cap \mathcal{P}'_s \neq \emptyset\},\\
  &\mathcal{R}' = \mathcal{R}'_l \cup \mathcal{R}'_{ip},\\
  &\mathcal{R}'_l = \{R \mid R \in \mathcal{R}_l \wedge \textsc{perc}(R) \cap \mathcal{P}'_s \neq \emptyset\},\\
  &\mathcal{R}'_{ip} = \{R \mid R \in \mathcal{R}_{ip} \wedge \textsc{rel}_{ip}(R) = \langle (U_1, P_1):O_1, \ldots, (U_k, P_k):O_k \rangle \wedge\\
  &\qquad \forall i \in \{1, \ldots, k\}\ (P_i \in \mathcal{P}'_s)\},\\
  &\mathcal{A}' = \{A \mid A \in \mathcal{A} \wedge \exists X \in \mathcal{O}' \cup \mathcal{R}'\ \exists P \in \mathcal{P}'\ \exists D \in \mathcal{D}\ \textsc{att}(X, P)[A] = D\},\\
  &\mathcal{U}' = \{U \mid U \in \mathcal{U} \wedge \exists R \in \mathcal{R}'_l\ \exists P \in \mathcal{P}'_s\ \exists O \in \mathcal{O}'\ (\textsc{rel}_l(R, P)[U] = O)\} \cup\\
  &\qquad \{U \mid U \in \mathcal{U} \wedge \exists R \in \mathcal{R}'_{ip}\ \exists P \in \mathcal{P}'_s\ \exists O \in \mathcal{O}'\ (\textsc{rel}_{ip}(R)[(U, P)] = O)\},\\
  &\mathcal{D}' = \{D \mid D \in \mathcal{D} \wedge \exists X \in \mathcal{O}' \cup \mathcal{R}'\ \exists P \in \mathcal{P}'\ \exists A \in \mathcal{A}'\ (\textsc{att}(X, P)[A] = D)\}.
  \end{aligned}$$

- $\textsc{perc}'$ is the total function that maps each object or relationship type symbol $X \in \mathcal{O}' \cup \mathcal{R}'$ to a nonempty set of perceptions defined by:

  $\forall X \in \mathcal{O}' \cup \mathcal{R}'\ (\textsc{perc}'(X) = \textsc{perc}(X) \cap \mathcal{P}')$.

- $\textsc{rel}'_l$ is the total function that maps a couple made up of a local relationship type symbol R in $\mathcal{R}'_l$ and a perception symbol $P \in \textsc{perc}'(R)$ to an $\mathcal{U}'$-labeled tuple over $\mathcal{O}'$ defined by:

  $\forall R \in \mathcal{R}'_l\ \forall P \in \textsc{perc}'(R)\ (\textsc{rel}'_l(R, P) = \textsc{rel}_l(R, P))$.

- $\textsc{rel}'_{ip}$ is the total function that maps an interperception relationship type symbol R in $\mathcal{R}'_{ip}$ to an $\mathcal{U}' \times \mathcal{P}'_s$-labeled tuple over $\mathcal{O}'$ defined by:

  $\forall R \in \mathcal{R}'_{ip}\ (\textsc{rel}'_{ip}(R) = \textsc{rel}_{ip}(R))$.

- $\textsc{att}'$ is the partial function that maps a couple made up of an object or relationship type symbol X in $\mathcal{O}' \cup \mathcal{R}'$ and a perception symbol $P \in \textsc{perc}'(X)$ to an $\mathcal{A}'$-labeled tuple over $\mathcal{D}'$ defined by:

  $\forall X \in \mathcal{O}' \cup \mathcal{R}'\ \forall P \in \textsc{perc}'(X)\ (\textsc{att}'(X, P) = \textsc{att}(X, P))$.

- CARD'_l is the partial function $\mathcal{O}' \times \mathcal{R}'_l \times \mathcal{U}' \times \mathcal{P}'_s \rightarrow \mathbb{N} \times (\mathbb{N} \cup \{\infty\})$ defined by:

  $\forall O \in \mathcal{O}' \; \forall R \in \mathcal{R}'_l \; \forall U \in \mathcal{U}' \; \forall P \in \text{PERC}'(R) \; (\text{REL}'_l(R,P)[U] = O \Rightarrow$
  $(\text{CARD}'_l(O,U,R,P) = \text{CARD}_l(O,U,R,P)))$.

- CARD'_{ip} is the partial function $\mathcal{O}' \times \mathcal{R}'_{ip} \times \mathcal{U}' \times \mathcal{P}'_s \rightarrow \mathbb{N} \times (\mathbb{N} \cup \{\infty\})$ defined by:

  $\forall O \in \mathcal{O}' \; \forall R \in \mathcal{R}'_{ip} \; \forall U \in \mathcal{U}' \; \forall P \in \mathcal{P}'_s \; (\text{REL}'_{ip}(R)[U,P] = O \Rightarrow$
  $(\text{CARD}'_{ip}(O,U,R,P) = \text{CARD}_{ip}(O,U,R,P)))$.

- ISA'_l is the ternary relation $\text{ISA}'_l \subseteq (\mathcal{O}' \times \mathcal{O}' \times \mathcal{P}'_s) \cup (\mathcal{R}'_l \times \mathcal{R}'_l \times \mathcal{P}'_s)$ defined by:

  $\forall X_1, X_2 \in \mathcal{O}' \cup \mathcal{R}'_l \; \forall P \in \mathcal{P}'_s \; (\text{ISA}'_l(X_1, X_2, P) \Leftrightarrow \text{ISA}_l(X_1, X_2, P))$.

- ISA'_{ip} is the quaternary relation $\text{ISA}'_{ip} \subseteq (\mathcal{O}' \times \mathcal{P}'_s \times \mathcal{O}' \times \mathcal{P}'_s) \cup (\mathcal{R}'_l \times \mathcal{P}'_s \times \mathcal{R}'_l \times \mathcal{P}'_s)$ defined by:

  $\forall X_1, X_2 \in \mathcal{O}' \cup \mathcal{R}'_l \; \forall P_1, P_2 \in \mathcal{P}'_s$
  $(\text{ISA}'_{ip}(X_1, P_1, X_2, P_2) \Leftrightarrow \text{ISA}_{ip}(X_1, P_1, X_2, P_2))$.

- OVLP'_l is the ternary relation $\text{OVLP}'_l \subseteq (\mathcal{O}' \times \mathcal{O}' \times \mathcal{P}'_s) \cup (\mathcal{R}'_l \times \mathcal{R}'_l \times \mathcal{P}'_s)$ defined by:

  $\forall X_1, X_2 \in \mathcal{O}' \cup \mathcal{R}'_l \; \forall P \in \mathcal{P}'_s \; (\text{OVLP}'_l(X_1, X_2, P) \Leftrightarrow \text{OVLP}_l(X_1, X_2, P))$.

- OVLP'_{ip} is the quaternary relation $\text{OVLP}'_{ip} \subseteq (\mathcal{O}' \times \mathcal{P}'_s \times \mathcal{O}' \times \mathcal{P}'_s) \cup (\mathcal{R}'_l \times \mathcal{P}'_s \times \mathcal{R}'_l \times \mathcal{P}'_s)$ defined by:

  $\forall X_1, X_2 \in \mathcal{O}' \cup \mathcal{R}'_l \; \forall P \in \mathcal{P}'_s \; (\text{OVLP}'_{ip}(X_1, P_1, X_2, P_2) \Leftrightarrow \text{OVLP}_{ip}(X_1, P_1, X_2, P_2))$.

- KEY' is the ternary relation $\text{KEY}' \subseteq (\mathcal{O}' \cup \mathcal{R}') \times \mathcal{P}' \times 2^{\mathcal{A}'}$ defined by:

  $\forall X \in \mathcal{O}' \cup \mathcal{R}' \; \forall P \in \mathcal{P}' \; \forall K \in 2^{\mathcal{A}'} \; (\text{KEY}'(X,P,K) \Leftrightarrow \text{KEY}(X,P,K))$.

The state $\mathcal{B}'$ of the perception is the tuple $\mathcal{B}' = (\Delta^{\mathcal{B}'}_{\mathcal{O}'} \cup \Delta^{\mathcal{B}'}_{\mathcal{R}'} \cup \Delta^{\mathcal{B}'}_{\mathcal{D}'}, \cdot^{\mathcal{B}'(P)})$ defined by:

- The function $\cdot^{\mathcal{B}'(P)}$, that, for some $P \in \mathcal{P}'$, maps:
  - Every domain symbol $D_i \in \mathcal{D}'$, for any perception $P \in \mathcal{P}'$, to the set $D_i^{\mathcal{B}'(P)} = \Delta^{\mathcal{B}}_{D_i}$;
  - Every object type symbol $O \in \mathcal{O}'$, for any of its perceptions $P \in \text{PERC}'(O)$, to the set $O^{\mathcal{B}'(P)} = O^{\mathcal{B}(P)}$;
  - Every relationship type symbol $R \in \mathcal{R}'$, for any of its perceptions $P \in \text{PERC}'(R)$, to the set $R^{\mathcal{B}'(P)} = R^{\mathcal{B}(P)}$);
  - Every attribute symbol $A \in \mathcal{A}'$, for a perception $P \in \mathcal{P}'$, to the set
    $A^{\mathcal{B}'(P)} = \{\langle x, a\rangle \mid \langle x, a\rangle \in A^{\mathcal{B}(P)} \wedge \exists X \in \mathcal{O}' \cup \mathcal{R}' \; \exists D \in \mathcal{D}'$
    $(P \in \text{PERC}'(X) \wedge \text{ATT}'(X,P)[A] = D \wedge x \in \text{PERC}'(X,P)\})$.

- $\Delta^{\mathcal{B}'}_{\mathcal{O}'} = \bigcup_{O \in \mathcal{O}' P \in \text{PERC}'(O)} O^{\mathcal{B}'(P)}$,
- $\Delta^{\mathcal{B}'}_{\mathcal{R}'} = \bigcup_{R \in \mathcal{R}' P \in \text{PERC}'(R)} R^{\mathcal{B}'(P)}$,
- $\Delta^{\mathcal{B}'}_{D'} = \bigcup_{D_i \in \mathcal{D}'} D_i^{\mathcal{B}'(P)}$. ◇

Proposition 3. If the set of perception symbols $\mathcal{P}'_s$ contains only one perception, the multiperception database $(\Sigma', \mathcal{B}')$ reduces to a SimpleMads database without perceptions. Indeed, the set $\mathcal{R}'_{ip}$ is empty, as well as the relations ISA'_{ip}, OVLP'_{ip}, REL'_{ip}, and CARD'_{ip}.

Theorem 1. The state $\mathcal{B}'$ of a perception is consistent, i.e., it satisfies all the constraints of the schema Σ'.

When the system receives an OpenDatabase(dbName, myView) command, it performs the following process: it matches the perceptions in myView with the set of perceptions of each object and relationship type of the database in order to determine which object and relationship types (with which properties and which populations) belong to the perception myView. Any element that belongs to at least one of the perceptions in myView belongs to the (composite) perception myView. Obviously, the myView representation of an object type includes all the representations of the attributes that belong to at least one of the perceptions in myView. If myView is a composite perception, this process may select several representations for the same attribute. Local relationship types follow the same selection process. However, whenever myView is a composite perception, the system has to perform an additional selection step to complete the definition of the myView perception: it has to look for interperception relationship types eligible for the given composite perception. The eligible interperception relationship types are those where all roles and linked object types belong to myView. The myView representation of the relationship type is made up of its selected roles and all the representations of all its attributes and semantics that belong to at least one of the perceptions in myView.

Let us refer to the database of Fig. 5.3 for an example of accessing an interperception relationship type. In order to know on which kind of soil – an attribute of GeologicalUnit – a specific vineyard is located, the user query has to go through the relationship type LocatedIn which does not belong to a simple perception. Thus, the user must open the database with the composite perception $(\mathsf{P}_\mathsf{m} + \mathsf{P}_\mathsf{g})$ for querying the LocatedIn relationship type.

When querying the database with a composite perception, users get for each query a multiperception answer, i.e., a set of answers, one per perception. The component answers are linked together by the fact that all representations describing the same object (respectively, relationship) instance are identified by the same system-defined identifier, oid (respectively, rid). For example, let us refer to the database of Fig. 5.6 and assume a user with the composite perception $(\mathsf{P}_1 + \mathsf{P}_2)$ who asks the following query: "Give me all animals". The answer will be:

$$\mathsf{P}_1 : \mathsf{a}_1, \mathsf{a}_2, \mathsf{a}_3, \mathsf{h}_1, \mathsf{h}_2$$
$$\mathsf{P}_2 : \mathsf{a}_1, \mathsf{a}_2, \mathsf{a}_3$$

Loading and updating data in a multiperception database may be done collaboratively by several users with different perceptions. An object (or relationship) instance that has several representations, say for perceptions $\mathsf{p}_1, \mathsf{p}_2, \ldots$, and p_k, may either be inserted (or deleted) in two ways:

- A user with the composite perception ($p_1 + p_2 + \ldots + p_k$) may insert (or delete) the whole instance with all its representations in a single operation; or
- The insertion (or deletion) is done by a sequence of operations: For each simple perception of the set $\{p_1, p_2, \ldots, p_k\}$, a user with this perception inserts (or deletes) the corresponding representation. When processing the first insert operation, the DBMS creates a new instance with a new oid and a unique representation. Each following insert operation adds a representation to the existing instance (there is no oid creation).

For example, adding in the Wine object type of Fig. 5.4 a new wine instance, say Clos Vougeot 2004, with its two representations may be done by a user with the composite perception ($P_m + P_e$) by giving the data for both representations as follows:

```
p = insertObject(Wine, {Pm, Pe}(
/* attributes common to perceptions Pm and Pe */
name = 'Clos Vougeot', year = 2004, ...
/* attributes specific to perception Pm */
description.atPerception(Pm) =
    'Sourced from old vines, the soft finish with silky tannins ...',
degree.atPerception(Pm) = 12.25, ...
/* attributes specific to perception Pe */
description.atPerception(Pe) =
    'Full of dark berry fruits on the nose, the palate depth shows ...',
...
degree.atPerception(Pe) = 12, ... ))
```

Alternatively, it can be done in two steps, e.g., a user of perception P_m inserting the P_m representation as in:

```
p = insertObject(Wine, {Pm}(
/* attributes of perception Pm */
name = 'Clos Vougeot', year = 2004, ...
description = 'Sourced from old vines, the soft finish with silky tannins
...',
degree = 12.25, ... ) )
```

and later a user of perception P_e inserting the P_e representation for the same Clos Vougeot 2004 instance as in:

```
p = select [name = 'Clos Vougeot' ∧ year = 2004 ] Wine ;
addObjectRepresentation(Wine, p, {Pe}(
/* attributes of perception Pe */
name = 'Clos Vougeot', year = 2004, ...
description = 'Full of dark berry fruits on the nose, the palate depth
shows ...', ...
degree = 12, ... ))
```

As can be seen, users willing to separately (i.e., perception per perception) create different representations for the same instance of a multiperception type have to agree on using the same key. This key must have a unique representation common to all perceptions. In the Wine object type, the common key is made up of two attributes, name and year. In the above example, the user must first obtain the identifier of the Clos Vougeot 2004 instance with a select operation in order to be able to add a representation to that instance.

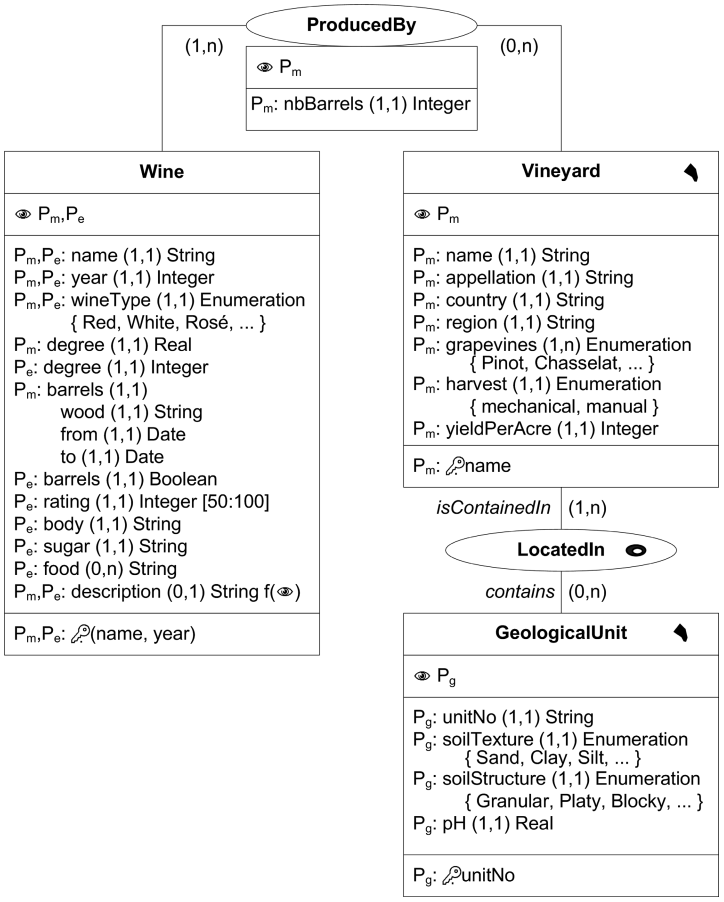
The existence of a relationship instance depends upon the one of the objects it links. In Entity-Relationship data models pending roles of relationships are prohibited. Thus, inserting or accessing a relationship instance, requires having access to the relationship type and to the linked object instances. Hence, inserting and accessing an instance of an interperception relationship type is only possible through a composite perception that contains all the perceptions of the linked object types. On the other hand, local relationship types that have several representations may, like object types, be inserted or deleted either by a unique operation on a composite perception that covers all the representations, or by a sequence of operations on simple perceptions. For example, the relationship type ProducedBy of Fig. 5.5 is local, i.e., any two instances, one of Wine and one of Vineyard, linked by a ProducedBy instance belong to the same perception as the ProducedBy instance. Let us assume that the two representations of ProducedBy differ by having different sets of attributes, e.g., number of bottles produced for P_m and vintage description for P_e. Then inserting a new instance of ProducedBy linking the Wine Clos Vougeot 2004 with oid p to the Vineyard Vigne du Clos Vougeot with oid q can be done either by one insert operation with perception $(P_m + P_e)$ as in:

```
insertRelationship(ProducedBy, {Pm, Pe}, Wine: p, Vineyard: q ) (
/* attributes of Pm */
numberOfBottles = 3500, ...
/* attributes of Pe */
description = 'The 2004 vintage had moderate rainfall in the winter
...', ... )
```

or by two insert operations, one with perception P_m and one with perception P_e.

5.8 Mapping into the Relational Model

For a database design based on Mads to be operational, we have defined an implementation approach that automatically transforms a Mads schema into an equivalent logical schema in the relational or object-relational data model, which can then be loaded into a commercial DBMS. The approach was materialized as a CASE tool, whose detailed description can be found in [10, 11]. This section discusses the general principles of that translation using the example schema in Fig. 5.9, which is an enriched version of the schema in Fig. 5.3 with all the attributes and perceptions shown. Our main intention is to show how multirepresentation features are conveyed into the logical schema.



ProducedBy
(1,n)
(0,n)
Pm
Pm: nbBarrels (1,1) Integer
Wine
Pm,Pe
Pm,Pe: name (1,1) String
Pm,Pe: year (1,1) Integer
Pm,Pe: wineType (1,1) Enumeration { Red, White, Rosé, ... }
Pm: degree (1,1) Real
Pe: degree (1,1) Integer
Pm: barrels (1,1) wood (1,1) String from (1,1) Date to (1,1) Date
Pe: barrels (1,1) Boolean
Pe: rating (1,1) Integer [50:100]
Pe: body (1,1) String
Pe: sugar (1,1) String
Pe: food (0,n) String
Pm,Pe: description (0,1) String f()
Pm,Pe: (name, year)
Vineyard
Pm
Pm: name (1,1) String
Pm: appellation (1,1) String
Pm: country (1,1) String
Pm: region (1,1) String
Pm: grapevines (1,n) Enumeration { Pinot, Chasselat, ... }
Pm: harvest (1,1) Enumeration { mechanical, manual }
Pm: yieldPerAcre (1,1) Integer
Pm: name
isContainedIn (1,n)
LocatedIn
contains (0,n)
GeologicalUnit
Pg
Pg: unitNo (1,1) String
Pg: soilTexture (1,1) Enumeration { Sand, Clay, Silt, ... }
Pg: soilStructure (1,1) Enumeration { Granular, Platy, Blocky, ... }
Pg: pH (1,1) Real
Pg: unitNo

Fig. 5.9. A detailed version of the schema of Fig. 5.3

Logical models target easiness and efficiency of implementation. They consequently support less sophisticated and poorer data structures than those of conceptual models. Therefore, when translating a conceptual schema into a logical schema, the critical issue is to avoid or at least limit the semantic loss due to the poorer expression power of logical data models. Usually, high-level features of conceptual models are translated into a combination of logical-level features, the combination aiming at filling the gap between the conceptual and logical constructs and minimize the semantic loss.

Let us illustrate this using the example schema from Fig. 5.9. Remark that Fig. 5.9 uses the same visual presentation as Fig. 5.4: all perceptions are merged, while Fig. 5.3 uses a visual presentation that separates the perceptions. Still the semantics conveyed by these two visual presentations is the same. For the translation into the relational model, basically there are two ways that are quite similar to the two

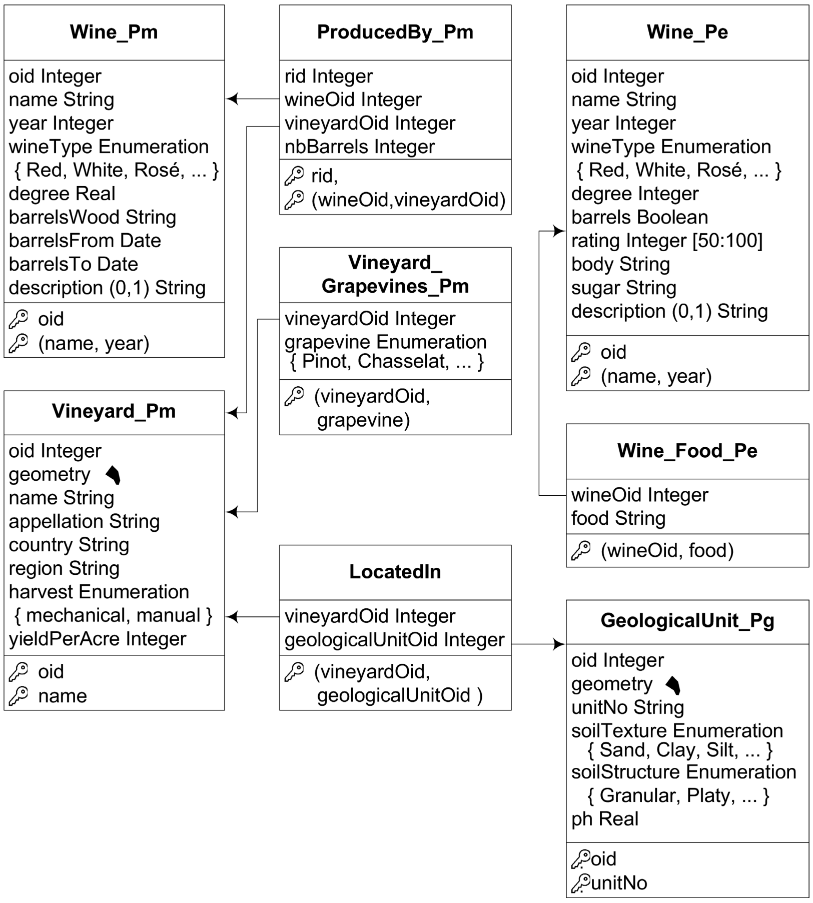


Fig. 5.10. Relational implementation of the schema of Fig. 5.9

visual presentations. These two ways generate relational schemas that are different but convey the same semantics. The difference between these two ways of translating is whether to create for each multi-perception object (and relationship) type a unique relational table containing all the attributes from the various perceptions, or several tables, one table for each perception that contains only the attributes defined for this perception. The first solution boils down to translating the multiperception schemas as presented with all perceptions merged, while the second solution to translating the multiperception schemas as presented with each perception on its own. The first solution generates tuples with NULL values each time that an object does not belong to all the perceptions defined for its object type. Here, we present the second solution, one table per object type and per perception. The translation algorithm consists in 1) applying to each perception the classic translation algorithm from the Entity

Relationship model without perception to the relational model, and 2) implementing each interperception relationship type by a relational table.

The result of the translation of the schema of Fig. 5.9 into a relational schema is shown Fig. 5.10. The first rule we used is: For each perception and for each of its object type, generate one primary table per perception[4]. In the example, as Wine belongs to two perceptions, its translation generates the two relational tables Wine_Pm and Wine_Pe, each one holding the monovalued attributes of perceptions P_m and P_e, respectively. The second rule states that composite attributes, such as barrels in P_m are replaced by their component attributes. This rule leads to a semantic loss (the composite attribute itself is lost), but the loss is in the label, the attribute values are preserved. The third rule is the traditional one that translates multivalued attributes by generating an additional table. In our running example, for perception P_e, the translation of the multivalued attribute food generates the table Wine_Food_Pe and the translation of the multivalued attribute grapevines for perception P_m generates the table Vineyard_Grapevines_Pm.

The relational representation of Wine does not make any difference between the attributes that are identical to both perceptions, such as name, year, and wineType, and the attributes whose value is perception dependent, such as description. Yet, in the conceptual specification, the values of the former attributes is shared by the two perceptions (i.e., the value is always the same in the two perceptions), while the values of the latter, description, are independent one from the other in the two perceptions. To prevent this semantic loss, the translation generates triggers (to be loaded into the target DBMS) to ensure that when users update an instance of, e.g., Wine_Pm, the updated values of name, year, and wineType (but not description) are propagated to the corresponding instance of Wine_Pe. Translation of local relationship types follows the same rule: one primary relational table per perception and per relationship type.

Like in traditional databases, roles of relationship types are translated into external keys. Lastly, each interperception relationship type, e.g LocatedIn, is translated into a relational table, exactly like for any classic Entity Relationship model.

Identifiers (oids and rids) simplify the translation of is-a and overlapping links (whether local or interperception). Consider again Fig. 5.7 where there is an interperception link between Wine in perception P_e and the three disjoint object types RedWine, WhiteWine, and RoseWine in perception P_e. In this case, the relational representation will include the tables Wine_Pm, RedWine_Pe, WhiteWine_Pe, and RoseWine_Pe, all of them with an attribute oid. The is-a relationship will be implemented by referential integrity contraints between each of the three tables in perception P_e and the table in perception P_m.

The identifiers help also to link several representations of the same instance. For example, if in Fig. 5.10 a wine has both representations P_m and P_e, the same oid value will be found in tables Wine_Pm and Wine_Pe. For example the following query retrieves the P_m representation of the wine "Zifandel Clos Marie" 2000:

[4] The primary table is the one holding all monovalued attributes of the object type.

```
SELECT * FROM Wine_Pm
WHERE Wine_Pm.name="Zifandel Clos Marie" AND Wine_Pm.year=2000
```

Similarly, the following query retrieves all representations of the same wine:

```
SELECT * FROM Wine_Pm FULL OUTER JOIN Wine_Pe
    ON Wine_Pm.oid=Wine_Pe.oid
WHERE Wine_Pm.name="Zifandel Clos Marie" AND Wine_Pm.year=2000
```

The translation of the Mads multiperception schema is completed by describing in the data dictionary of the relational database the set of simple perceptions of the schema and for each simple perception the set of tables that belong to that perception. This information can be organized as one table:

```
SimplePerceptionTables (perceptionId, tableName)
```

Another table is required to store the definition of interperception relationship types and the associated object types:

```
InterPerceptionRelationships (relationshipTable, objectTable)
    objectTable REFERENCES SimplePerceptionTables
```

The content of these two tables for the database of Fig. 5.10 is given in Fig. 5.11. These tables are used by the system when a user begins a working session by opening the multiperception database with either a simple or a composite perception. For example, a user opens the Mads database of Fig. 5.9 – let us call WineDB this database – by issuing the following command:

```
openDatabase (WineDB, Pm)
```

The system, after looking at the SimplePerceptionTables table, will give to the user the access rights to the Wine_Pm, Vineyard_Pm, and ProducedBy_Pm tables. On the other hand, if a user issues:

```
openDatabase (WineDB, (Pm+Pg))
```

the system will look for the tables to which the user will be given access rights by searching in the SimplePerceptionTables and InterPerceptionRelationships tables. The resulting list of tables will be: Wine_Pm, Vineyard_Pm, ProducedBy_Pm, GeologicalUnit_Pg, and LocatedIn.

SimplePerceptionTables

perceptionId	tableName
Pm	Wine_Pm
Pm	Vineyard_Pm
Pm	ProducedBy_Pm
Pe	Wine_Pe
Pg	GeologicalUnit_Pg

InterPerceptionRelationships

relationshipTable	objectTable
LocatedIn	Wine_Pm
LocatedIn	GeologicalUnit_Pg

Fig. 5.11. The data dictionary of the perceptions of the database in Fig. 5.10

5.9 Related Work

This section compares the perception mechanism in Mads to other approaches that similarly aim at supporting multiple perspectives on the same information repository, database or ontology, be it for contextualization or modularization purposes.

On Supporting Multiple Perspectives

As the concept of perspective is subject to a variety of interpretations, a variety of mechanisms have been developed, first in the database domain, to meaningfully partition an information system into subsets defined for different purposes or characterized by different properties. Views and versions are available in commercial DBMS. Distributed data solutions (e.g., federated databases, multidatabases) have been defined to support modules and are therefore related to the modularization topic of this book, but have had an impact only in the research community, except for the simplest category (distributed databases) that only implies managing multiple storage systems. The view mechanism is the most widespread and its use is routine work for database administrators. It has also been defined for ontologies, as shown in Part II of this book. A view is an on-demand personalized data structure (at the logical level) built from the underlying data structures implemented in the database. As discussed in detail in the introductory section 5.1 of this chapter, views provide poorer functionality than Mads perception concept. Basically, the scope of a view is to provide an application-specific perspective on an object type, while the scope of a perception is to provide an application-specific (conceptual) perspective on the whole database. Moreover, views are mostly intended for data retrieval. Updates can be performed onto a view only if the view derivation process satisfies some quite restrictive rules. For example, one rule states that the columns being modified in the view must directly reference the underlying data in the table columns, and thus, e.g., cannot be derived through an aggregate function, cannot be computed from an expression that uses other columns, or cannot be formed by using set operators such as union, difference or intersection. The reason for these restrictive rules is that the system must be able to unambiguously translate modifications in the view into modifications in the base tables from which the view is derived. In case of ambiguity, an update of a view element can nevertheless be allowed if INSTEAD OF triggers have been manually and explicitly defined by the database administrator to state how any given modification to the data in the view is to be translated into modifications to the underlying base tables. Perceptions, instead, are meant to fully support application-specific data management, not just retrieval. They are therefore updatable, unless specific application constraints are defined to restrict updatability.

Versioning, as the name says, is a mechanism specifically designed to support change management. It allows managing an ordered graph of versions of the same element (document, object, database, ...). Its main functionality is to enable backtracking to previous versions of an element and to retrieve a consistent set of versions of parts of a composite element (typically sections in a document) when these parts have evolved in a-synchronized way in a collaborative environment. The perspective provided by a version is alike a temporal perspective, but instead of looking at the state of affairs at a certain instant in time it looks at the state of affairs at a certain moment of an evolution path. Although tagging with a version identifier can be seen, at least to some extent, as similar to tagging with a perception identifier, the two approaches rely on fundamentally different paradigms. Versions offer successive images of an evolving element, while perceptions offer complementary images of an element taken at the same moment in time. It would not be wise to confuse users (and the system) by offering versioning concepts to support perceptions.

Closer to the Mads perception concept is the contextual module concept proposed by Mylopoulos and Motschnig [7, 8]. The authors propose a generic abstract model, independent of any specific information model, which supports modules, called contexts. They specify basic rules for defining an information model with modules. An example is the rule stating that elements belonging to several modules should be allowed to have a specific name local to each module. Another rule states that whenever two modules share some elements, they should agree on the propagation of their updates. The Mads approach was defined independently of that work, but its principles are very much in line with the work and its results confirm and refine the ideas in [7, 8].

Cyc Microtheories

Let us now turn our attention to ontologies and look at Mads as an approach to create modular ontologies, with each module representing a specific perception on the existing data. According to this view, Mads perception modules are very similar to Cyc microtheories concept and mechanism. Indeed, both Mads and Cyc build on the idea that an ontology and its modules are simultaneously defined. In other words, when creating an element (at the type or instance level) to be added to an ontology, the creator also specifies to which module(s) the element is to be added. Given the strong similarity between Mads and Cyc, we present here Cyc in more detail and then compare it with Mads.

Cyc is a large, still developing, modular knowledge base that was created in the late 80ies. Its goal is to "cover all common sense" for supporting reasoning in a variety of domains [6]. Examples of potential applications are text understanding, knowledge management, question answering, expert systems, intelligent search, semantic integration. The authors of Cyc had, since the beginning, a pragmatic approach. They wanted to build something that will work and be useful . . . and indeed they succeeded. The Cyc engineers have now entered enough basic knowledge in Cyc for automatic tools to being able to pursue autonomously the same goal, increasing Cyc knowledge base by browsing web pages, understanding and checking the content of these pages, and then adding it to the Cyc knowledge base. The Cyc knowledge base reaches beyond 3 millions assertions, 300'000 terms, and 15'000 predicates

Cyc is described in a specific logic, which mainly is first order logic with higher order extensions that support quantification over predicates. The Cyc formalism, called CycL, for Cyc language, has a syntax that is similar to the one of LISP. Cyc knowledge is described through sentences that are built up from individual items, called individuals, like Paris and Europe, concepts that group sets of individuals, like City or UrbanArea, n-ary predicates, functions, variables, logical connectors, and quantifiers. The most important predicates are isa and genls, which are two generic binary predicates expressing membership and generalization/specialization, respectively. For instance, the (membership) predicate: *(isa Paris City)* expresses the fact that Paris is a City. And the (generalization/specialization) predicate *(genls City UrbanArea)* expresses the fact that every city is also an urban area. Finally, the following CycL sentence: *(implies (and (isa ?ELT ?SUBSET) (genls ?SUBSET ?SUPERSET)) (isa ?ELT ?SUPERSET))* is a rule which is part of the upper knowledge of Cyc. It defines part of the semantics of the genls predicate. It states that if an element (?ELT) is a member of a collection (?SUBSET) and this collection is a specialization of another collection (?SUPERSET), then the element is also a member of the second collection.

Cyc started as an unstructured knowledge base, i.e., a set of assertions. Later it was seen as covering the three traditional levels of ontologies: 1) Upper ontology: general knowledge (top level ontology); 2) Middle ontology: not universal, but commonly used knowledge, and 3) Lower ontology: knowledge specific to a peculiar domain (domain ontology). As the knowledge base grew, editing it became more and more difficult. Cyc engineers decided to organize Cyc into small independent modules, called *microtheories* or *contexts* [5]. Each microtheory is supposed to describe the knowledge of a specific domain. For instance, MathMt is a microtheory that contains mathematical knowledge. Microtheories are organized in an inheritance hierarchy that supports multiple inheritance. If a microtheory is a child of another microtheory, its knowledge (a set of CycL assertions) is made up of its own (local) knowledge plus the knowledge of its parent microtheory and recursively all the knowledge of their parents. Obviously, the local knowledge of a microtheory must be consistent with all its inherited knowledge. For instance the GeometryMt microtheory, the microtheory describing geometry, is a child of the MathMt microtheory. A query on GeometryMt will involve all assertions of MathMt, and recursively. Given two microtheories such that neither one inherits from the other one, their local knowledge can be inconsistent. For instance, a microtheory can state that a human being is an animal that is a cousin of apes, while another microtheory will state that human beings are not animals. The partitioning of Cyc in microtheories allows the recording of different points of view which may result from different groups of users (e.g. liberalism as seen by republicans and as seen by democrats), different usages, different granularities (e.g. Lausanne in a world atlas and Lausanne in Swiss maps), different time frames (e.g. Rome today and Rome 2000 years ago)... Queries are local to a microtheory. When users want to query Cyc knowledge base they have to send their query to a specific microtheory. Similarly, when Cyc engineers want to add some knowledge (i.e. some Cyc assertions) they have to define to which microtheory this knowledge will be added. When creating a new microtheory, they have to specify where the new microtheory will be inserted in the inheritance hierarchy. Cyc provides tools for helping users in these tasks.

Comparing Cyc and Mads

Let us now compare the modular ontology, Cyc, and the modular database approach, Mads. The two main similarities between Cyc and Mads are that 1/ they both are modular repositories (a modular knowledge base for Cyc and a modular database for Mads), i.e. repositories purposely created as composed of modules, and 2/ they both want to be running systems: Cyc is a large running ontology, Mads is a prototype running onto the Oracle DBMS and has been tested with several real life applications. However, even in their commonalities, Cyc and Mads differ. For the first point, Cyc is more dynamic than Mads, as new modules can easily be added at any time. On the other hand, Mads, being a database, is more static: The basic modules of Mads (called simple modules in this Chapter) are defined during the design of the schema of the database. For the second point, Cyc has been in use since many years, while Mads is still a research prototype.

The main difference between Cyc and Mads is how modules are related together. Cyc modules are disjoint and are related by an inheritance hierarchy that allows a module to inherit the whole content of one or several other modules (and recursively the whole content of all their ancestors in the hierarchy). In Mads modules may overlap, and the sharing unit is the class instance: Two modules may share one, several, or all the instances of a class. Moreover in Mads inter-modules relationships can relate an instance of a class of a module to an instance of a class of another module.

Another difference between Cyc and Mads, resulting from the fact that Mads supports instances shared by several modules, is that in Mads these shared instances may have a specific representation (specific structure and specific value) for each module.

A common choice of Cyc and Mads is that both systems require that their users specify which modules they want to access. But as Mads modules are inter-related and not Cyc ones, Cyc users can access only one module (including its ancestor modules) at a time, while Mads users can access one or several modules at a time. This multi-module access allows Mads users to get simultaneously several descriptions (called representations in Mads) for the same entities.

Creating New Modules

Cyc and Mads are representative of the approach “Creating the ontology and its modules simultaneously”. But most of actual research on modularity in ontologies follows different approaches, namely ontology partitioning, module extraction, and interconnection of existing ontologies (described in Parts II and III of this book). In particular, module extraction approaches allow the dynamic creation of new modules, whenever there is a need for a new module and without an impact on the already existing modules. The creation of a new module from a running database is also possible in Mads. The definition of an additional simple perception can be done anytime through a schema modification process. First, an identifier has to be specified for the new perception, and second, the definition of the database schema has to be revisited to add the new perception identifier to the set of representations associated to the elements (schema and instances) the database administrator wants to see in the new perception. This extensional process has to be validated by checking the consistency rules that enforce a perception to obey the modeling constraints of a normal database. The other Mads mechanism that dynamically creates new perceptions is the composition of existing perceptions into a composite perception. This intensional process is prompted anytime a transaction uses the openDatabase command with a composite perception. The new composite perception remains a virtual one. It is not materialized. Nevertheless, the database administrator can anytime decide to materialize a composite perception, if required.

Distributed Modules

Let us now compare the constructs supported by the Mads data model to the ones supported by approaches that connect existing ontologies. Mads supports three kinds of links between perceptions/modules:

1. Whenever an object (or relationship) type is defined as having multiple representations, hence belonging to multiple perceptions, the shared type implicitly defines a link between these perceptions. At the type level the link is materialized by the fact that the shared type has the same name in the various perceptions, while at the instance level it is materialized by the fact that the shared instances bear the same oid (rid).
2. Two different object types belonging to two different perceptions, but representing at least partially the same real-world entities, may be linked by an interperception multi-instantiation link (is-a or overlap link).
3. Two different object types belonging to two different perceptions may be linked by an interperception relationship type.

The first and second kinds of interperception links, in the case where the populations of the linked types are one included in or equal to the other, are similar to the bridge rules of C-OWL, which allow relating two concepts that, in any interpretation, describe two sets of entities that are linked by an inclusion [3]. A difference between the Mads links and the bridge rules is that Mads first mechanism works even if the populations are disjoint, and Mads second mechanism works with included or overlapping populations. On the other hand, bridge rules are only intended for two concepts related by an inclusion (or an equality). Mads third mechanism is similar to the link property of $\mathcal{E}$–connections that allows relating two classes from disjoint modules (i.e., modules that describe disjoint parts of the world) by an inter-module role, called link property [4].

The main difference between Mads and approaches that connect existing ontologies is that Mads allows representing the same real-world phenomenon with representations that are quite dissimilar from each other. Two representations may have disjoint populations and still be two representations of the same object type. For instance, a perception of the Wine object type may describe only European wines while another perception may describe only American wines. As another example, in Fig. 5.4 the attribute barrels has two representations that are quite dissimilar, and still in Mads these two representations are related: Users querying the barrels attribute with the composite perception $(\mathsf{P}_\mathsf{m} + \mathsf{P}_\mathsf{e})$ get for each wine two values, one for each perception. This possibility of stating that several object types, several relationship types, or several attributes describe the same phenomenon, even if they are totally different, is – as far as we know – peculiar to Mads.

5.10 Conclusion

This chapter has described an approach to database modularization in terms of supporting multiple perceptions over a database and multiple representations of its elements. The new concepts and rules that form the approach are presented as embedded in the Mads conceptual data model. Perception features can thus be applied to the thematic as well to the spatio-temporal characteristics of a database. The chapter focused on discussing perceptions. A detailed description of other Mads features can be found elsewhere [10], namely including its concepts, how to use them in database modeling, and the operations to work with these concepts to create and maintain a multiperception database.

Defining data corresponding to a specific perception is equivalent to defining a module in a modular database. To this extent, the perception and module concepts are synonyms. This explains that the Mads approach and solution share many commonalities with the Cyc approach to modular ontologies. However, differences between the goals of Mads and Cyc induce different solutions. Cyc is a huge and still growing ontology where reuse is important. Therefore, the organization of Cyc modules is an inheritance hierarchy, while Mads modules are organized along a composition graph. Mads' goal is to provide different groups of users with different perceptions of the same database. Consequently, all Mads modules share the same interpretation domain, while each Cyc microtheory has its own. Moreover, in Mads the system is aware – and manages – the fact that the same real-world phenomenon is described by several representations. In Cyc two microtheories may contain different representations of the same phenomenon but Cyc ignores it. Users interested in modularizing a knowledge repository from its creation onwards should carefully analyze which goal they are trying to achieve to choose the most suitable solution.

The Mads model has been used in many real-world applications. For example, in a cartographic application at the French Mapping Agency (IGN) the multirepresentation features of the model were used for describing the representations of geographic objects at different levels of detail (i.e., resolution). This cartographic application also needed the interperception links to compare the different representations of real-world objects for validation purposes. In another application realized at Cemagref, a research center on risk management, perceptions were used to define different user profiles to obtain customized information from the same database. For example, information about natural risks, such as avalanches or landslides, is delivered to users depending on their profile: the general public obtains validated and less technical information with respect to risk experts.

Future work for the Mads model includes the extension of interperception links between constructs of different kinds, e.g., when the same phenomenon is represented as an object type in one perception and as an attribute or as a relationship type in another perception. In view of targeting semantic Web applications a formalization of the Mads model according to latest W3C standards would be needed. Unfortunately, spatio-temporal semantics is not supported by standard description logics. For this reason we are exploring a complementary approach consisting in using database technology (which somehow knows how to manage spatio-temporal data) to handle ontological data and services. As a first step, a prototype, called OntoMinD, has been developed to store large DL ontologies in an extended object-relational database system. The OntoMinD extension relies on the specification of a set of stored procedures that perform ontological reasoning on the TBox and ABox [1].

References

1. Al-Jadir, L., Parent, C., Spaccapietra, S.: OntoMind: Reasoning with Large DL Ontologies Stored in Relational Databases (2009) (in preparation)
2. Artale, A., Parent, C., Spaccapietra, S.: Evolving objects in temporal information systems. Annals of Mathematics and Artificial Intelligence 50(1–2), 5–38 (2007)
3. Bouquet, P., Giunchiglia, F., van Harmelen, A., Serafini, L., Stuckenschmidt, H.: Contextualizing ontologies. Journal of Web Semantics 1(4), 325–343 (2004)
4. Grau, B.C., Parsia, B., Sirin, E.: Ontology Integration Using $\mathcal{E}$-connections. In: Stuckenschmidt, H., Parent, C., Spaccapietra, S. (eds.) Modular Ontologies. LNCS, vol. 5445, pp. 293–320. Springer, Heidelberg (2009)
5. Cycorp. What is a context (2006), `http://www.cyc.com/cycdoc/course/what-is-a-context.html`
6. Lenat, D.B., Guha, R.V.: Building Large Knowledge-Based Systems. In: Representation and Inference in the Cyc Project. Addison-Wesley, Reading (1989)
7. Mylopoulos, J., Motschnig-Pitrig, R.: Partitioning information bases with contexts. In: Proceedings of the 3rd International Conference On Cooperative Information Systems, CoopIs 1995, pp. 44–54 (1995)
8. Motschnig-Pitrig, R.: A generic framework for modelling contexts and its applications. Data and Knowledge Engineering 32(2), 145–180 (2000)
9. Open Geospatial Consortium Inc. OpenGIS Implementation Specification for Geographic information – Simple feature access – Part 2: SQL option OGC 06-104r3, Version 1.2.0 (2006)
10. Parent, C., Spaccapietra, S., Zimányi, E.: Conceptual Modeling for Traditional and Spatio-Temporal Applications: The MADS Approach. Springer, Heidelberg (2006)
11. Parent, C., Spaccapietra, S., Zimányi, E.: The MurMur Project: Modeling and Querying Multi-Represented Spatio-Temporal Databases. Information Systems 31(8), 733–769 (2006)
12. Spaccapietra, S., Parent, C., Zimany, E.: Spatio-temporal and multirepresentation modeling for supporting active conceptual modeling of learning, ACM-L. In: Chen, P.P., Wong, L.Y. (eds.) ACM-L 2006. LNCS, vol. 4512, pp. 194–205. Springer, Heidelberg (2007)

Part II

Partitioning and Extraction of Modules

Introduction to Part II

In an ideal world, ontologies would be build in a modular way from the start thus showing the benefits well known from modularization in software engineering. The first part of this book contained an example of how this can be done for databases. Unfortunately, the field of ontology engineering has not yet developed comprehensive models and methods to fully support the development of modular ontologies. There are some approaches aiming at developing principles and formalisms in this direction, but they have not yet found their way into mainstream ontology engineering. As a result, existing ontologies are monolithic models without a clear internal structuring. As the size and complexity of these models can be quite significant (the NCI cancer ontology contains about 27.500 the Gene ontology about 22.000 concepts and the Formal Model of Anatomy (FMA) even 75.000 concepts). A viable way of handling such large models is to chop them up into manageable parts. This part of the book deals with approaches for this task of partitioning large ontologies into smaller parts.

As already discussed in chapter 3, there are various approaches to this problem that differ in the goal and the criteria used for determining modules in an ontology. In particular, we can distinguish between partitioning approaches that aim at splitting the entire ontology up into a set of possibly overlapping modules according to some criteria and module extraction approaches, whose goal in contrast is to extract a single module from a large ontology that contains relevant information for a given task. Concerning criteria for determining modules, we can roughly distinguish between logical and structural criteria - a more detailed typology of criteria is given in chapter 3. While logical criteria use the notion of logical consequence to determine what information should reside in the same module, normally with the goal of ensuring the completeness of local reasoning, structural approaches analyze the explicit definitions contained in an ontology with respect to dependencies between different elements in the ontology and try to group highly related elements in the same module. The process of determining strongly related elements is normally performed by converting the ontology into a graph structure and using graph algorithms for determining modules.

In this part of the book, a number of concrete approaches for determining modules of large ontologies are presented in more detail. The methods presented differ significantly in the way they approach the problem of modularizing an ontology thereby providing examples for the different goals and criteria mentioned above and discussed in chapter 3.

In chapter 6 Cuenca-Grau and others present an approach for partitioning ontologies based on logical criteria. This work is part of a research effort of the authors that was the first serious attempt to tackle the problem of partitioning description logic ontologies from a logical point of view and is still a cornerstone for theoretical work on modularity in ontologies. In this chapter the authors provide a logical definition of the notion of a module in terms of the completeness of reasoning with respect to a certain subset of the signature and provide some negative results concerning the computability of such modules for reasonably expressive languages. Further, the authors show that the notion of a conservative extension that is discussed in detail in chapter 2 can be used as a sound, but incomplete test for a module and present algorithms for performing this test.

The PATO System presented in chapter 7 takes a very different approach by completely relying on structural criteria for partitioning ontologies. The underlying partitioning method was one of the first methods being proposed for the partitioning task in relation to ontologies on the semantic web. In contrast to the logic-based approach, the partitioning method described in this chapter has the disadvantage that it cannot make any guarantees with respect to logical properties of the resulting modules, on the other hand, it has the advantage, that it can be applied to a wide range of models including simple taxonomies on which the logic-based approach fails to produce sensible results. Further, the method is designed in such a way that it can be adapted to different applications because the system allows to define and adapt the criteria used to determine modules. The corresponding mechanisms are explained in detail in the chapter.

The method described in section 7 was aimed at generality thereby sacrificing opportunities for of modularization for specific, highly relevant cases. Clearly focussing on ontologies specified using the web ontology language OWL is such a relevant case. While the PATO system was also designed for handling OWL ontologies, it does not deeply investigate the different modelling elements of OWL and their potential contribution to the determination of modules. This gap is filled by the work presented in chapter 8 where Seidenberg presents an approach for extracting a relevant module from an OWL ontology using an algorithm that traverses an OWL model following certain typical modelling constructs. The method returns part part of the ontology that is relevant with respect to a given class thus aiming at module extraction rather than partitioning. The chapter presents the method and shows a detailed evaluation in the area of medical ontologies.

A very similar approach to the problem of extracting the relevant part of an ontology is presented in chapter 9. In this work, which was originally done before the approach described in chapter 8, reduces the problem of module extraction to the problem of defining a specific view over the ontology at hand, thereby borrowing from concepts in the database area. The advantage of this approach is the ability to compose different definitions thereby creating new modules based on the definitions of other ones. This declarative approach to defining modules eases the investigation of formal properties of the resulting modules in terms of properties of certain operators defined over modules, an idea that has been used before to characterize ontologies and has currently been adopted for the analysis of semantic links between different ontologies, a topic that is treated in more details in part III of this book.

October 2008

Heiner Stuckenschmidt
Christine Parent
Stefano Spaccapietra

6

Extracting Modules from Ontologies: A Logic-Based Approach

Bernardo Cuenca Grau[1], Ian Horrocks[1], Yevgeny Kazakov[1], and Ulrike Sattler[2]

[1] University of Oxford, UK
[2] University of Manchester, UK

Summary. The ability to extract meaningful fragments from an ontology is essential for ontology reuse. We propose a definition of a module that guarantees to completely capture the meaning of a given set of terms, i.e., to include all axioms relevant to the meaning of these terms. We show that the problem of determining whether a subset of an ontology is a module for a given vocabulary is undecidable even for OWL DL. Given these negative results, we propose sufficient conditions for a for a fragment of an ontology to be a module. We propose an algorithm for computing modules based on those conditions and present our experimental results on a set of real-world ontologies of varying size and complexity.

6.1 Introduction

The design, maintenance, reuse, and integration of ontologies are complex tasks. Like software engineers, ontology engineers need to be supported by tools and methodologies that help them to minimize the introduction of errors, i.e., to ensure that ontologies are consistent and do not have unexpected consequences. In order to develop this support, important notions from software engineering, such as *module*, *black-box behavior*, and *controlled interaction*, need to be adapted.

For example, suppose that an ontology engineer is building an ontology about research projects, which specifies different types of projects according to the research topic they focus on. The ontology engineer in charge of the projects ontology may use terms such as Cystic_Fibrosis and Genetic_Disorder in his descriptions of medical research projects. The ontology engineer is an expert on research projects; he may be unfamiliar, however, with most of the topics the projects cover and, in particular, with the terms Cystic_Fibrosis and Genetic_Disorder. In order to complete the projects ontology with suitable definitions of these medical terms, he decides to reuse the knowledge about these subjects from a well-established medical ontology $\mathcal{Q}$.

The most straightforward way to reuse these concepts is to construct the logical union $\mathcal{P} \cup \mathcal{Q}$ of the axioms in $\mathcal{P}$ and $\mathcal{Q}$. This form of reuse is used frequently and can be achieved in OWL using the $\mathsf{owl:imports}$ construct. Well-established medical ontologies, such as NCI and SNOMED, are, however, typically very large, and importing the whole ontology would make the consequences of the additional information costly to compute and difficult for our ontology engineers (who are not medical experts) to understand. Thus, in practice, we need to extract a module that includes just the relevant information. Ideally, this module should be *as small as possible* while still *guaranteeing* to capture the meaning of the terms used; that is, when answering arbitrary queries against our projects ontology, importing the module would give us *exactly the same answers* as if we had imported the whole medical ontology. In this case, importing the module instead of the whole ontology will have no observable effect on our ontology—apart from allowing for more efficient reasoning.

Concerning the efficiency of reasoning, the time needed to process an ontology is often too high for ontology engineering, where fast response under changes in the ontology is required, or for deployment in applications, where fast response to queries is required. The ability to extract modules in the sense described above would address both these problems: it would allow us to identify a (hopefully small) part of the ontology that is affected by a given change or that is sufficient to answer a given query—and then to reason over this part only without losing any consequences.

The contributions of this paper are as follows:

1. We propose a definition of a *module* $\mathcal{Q}_1$ within a given ontology $\mathcal{Q}$ for a given vocabulary $\mathbf{S}$.
2. We take the above definition as a starting point, and investigate the problem of computing modules. We show that none of the reasonable variants of this problem is solvable in general already for rather restricted sub-languages of OWL DL. In fact, it is even not possible to determine whether a subset $\mathcal{Q}_1$ of an ontology $\mathcal{Q}$ is a module in $\mathcal{Q}$ for $\mathbf{S}$.
3. Given these negative results, we propose sufficient conditions for a fragment of an ontology to be a module—that is, if the fragment satisfies our conditions then we can guarantee that it is a module but not vice versa These conditions are based on the notion of locality of an ontology w.r.t. a signature, as first introduced in [4].
4. We propose an algorithm for computing locality-based modules.
5. We describe our implementation and present empirical results on a set of real-world ontologies of varying size and complexity. Using our syntactic approximation, we obtain modules that are small enough for reuse applications.

6.2 Preliminaries

In this section we introduce description logics (DLs) [2], a family of knowledge representation formalisms which underlie modern ontology languages, such as OWL DL [16]. A hierarchy of commonly-used description logics is summarized in Table 6.1.

The *syntax* of a description logic $\mathbb{L}$ is given by a signature and a set of constructors. A *signature* (or *vocabulary*) $\mathbf{Sg}$ of a DL is the (disjoint) union of countably infinite sets $\mathbf{C}$ of *atomic concepts* $(A, B, \dots)$ representing sets of elements, $\mathbf{R}$ of

Table 6.1. The hierarchy of standard description logics

	Constructors		Axioms [$\mathbf{Ax}$]		
	$\mathbf{Rol}$	Con	RBox	TBox	ABox
$\mathcal{EL}$	r	$\top, A, C_1 \sqcap C_2, \exists R.C$		$A \equiv C, C_1 \sqsubseteq C_2$	$a : C, r(a, b)$
$\mathcal{ALC}$	–||–	–||–, $\neg C$		–||–	–||–
$\mathcal{S}$	–||–	–||–	$\mathsf{Trans}(r)$	–||–	–||–
+ $\mathcal{I}$	r^-				
+ $\mathcal{H}$			$R_1 \sqsubseteq R_2$		
+ $\mathcal{F}$			$\mathsf{Funct}(R)$		
+ $\mathcal{N}$		$(\geqslant n\, S)$			
+ $\mathcal{Q}$		$(\geqslant n\, S.C)$			
+ $\mathcal{O}$		$\{i\}$			

Here $r \in \mathbf{R}$, $A \in \mathbf{C}$, $a, b \in \mathbf{I}$, $R_{(i)} \in \mathbf{Rol}$, $C_{(i)} \in \mathsf{Con}$, $n \geq 1$ and $S \in \mathbf{Rol}$ a simple role[10].

atomic roles $(r, s, \dots)$ representing binary relations between elements, and **I** of *individuals* $(a, b, c, \dots)$ representing constants. We assume the signature to be fixed for every DL.

Each DL provides *constructors* for defining the set $\mathbf{Rol}$ of (general) *roles* $(R, S, \dots)$, the set $\mathbf{Con}$ of (general) *concepts* $(C, D, \dots)$, and $\mathbf{Ax}$ of *axioms* $(\alpha, \beta, \dots)$ which includes the *role axioms* (RBox), *terminological axioms* (TBox) and *assertions* (ABox).

$\mathcal{EL}$ [1] is a simple DL which allows one to construct complex concepts using *conjunction* $C_1 \sqcap C_2$ and *existential restriction* $\exists R.C$ starting from atomic concepts A, roles R and the *top concept* $\top$. $\mathcal{EL}$ provides no role constructors and no role axioms; thus, each role R in $\mathcal{EL}$ is atomic. The TBox axioms of $\mathcal{EL}$ can be either *concept definitions* $A \equiv C$ or *general concept inclusion axioms* (GCIs) $C_1 \sqsubseteq C_2$. $\mathcal{EL}$ assertions are either *concept assertions* $a : C$ or *role assertions* $r(a, b)$. We assume the concept definition $A \equiv C$ is an abbreviation for two GCIs $A \sqsubseteq C$ and $C \sqsubseteq A$.

The basic description logic $\mathcal{ALC}$ [17] is obtained from $\mathcal{EL}$ by adding the *concept negation* constructor $\neg C$. We introduce some additional constructors as abbreviations: the *bottom concept* $\bot$ is a shortcut for $\neg\top$, the *concept disjunction* $C_1 \sqcup C_2$ stands for $\neg(\neg C_1 \sqcap \neg C_2)$, and the *value restriction* $\forall R.C$ stands for $\neg(\exists R.\neg C)$. In contrast to $\mathcal{EL}$, $\mathcal{ALC}$ can express *contradiction axioms* like $\top \sqsubseteq \bot$. The logic $\mathcal{S}$ is an extension of $\mathcal{ALC}$ where, additionally, some atomic roles can be declared to be *transitive* using a role axiom $\mathsf{Trans}(r)$.

Further extensions of DLs add features such as *inverse roles* r^- (indicated by appending a letter $\mathcal{I}$ to the name of the logic), *role inclusion axioms* (RIs) $R_1 \sqsubseteq R_2$ $(+\mathcal{H})$, *functional roles* $\mathsf{Funct}(R)$ $(+\mathcal{F})$, *number restrictions* $(\geqslant n\, S)$, with $n \geq 1$, $(+\mathcal{N})$, *qualified number restrictions* $(\geqslant n\, S.C)$, with $n \geq 1$, $(+\mathcal{Q})$[1], and *nominals* $\{a\}$ $(+\mathcal{O})$. Nominals allow for the construction of concepts representing a singleton

[1] The dual constructors $(\leqslant n\, S)$ and $(\leqslant n\, S.C)$ are abbreviations for $\neg(\geqslant n + 1\, S)$ and $\neg(\geqslant n + 1\, S.\neg C)$, respectively.

set $\{a\}$ (a *nominal* concept) from an individual a. These extensions can be used in different combinations; for example $\mathcal{ALCO}$ is an extension of $\mathcal{ALC}$ with nominals; $\mathcal{SHIQ}$ is an extension of $\mathcal{S}$ with role hierarchies, inverse roles and qualified number restrictions; and $\mathcal{SHOIQ}$ is the DL that uses all the constructors and axiom types we have presented.

Modern ontology languages, such as OWL, are based on description logics and, to a certain extent, are syntactic variants thereof. In particular, OWL DL corresponds to $\mathcal{SHOIN}$ [9]. In this paper, we assume an *ontology* $\mathcal{O}$ *based on a description logic* $\mathbb{L}$ to be a finite set of axioms in $\mathbb{L}$. The *signature of an ontology* $\mathcal{O}$ (*of an axiom* α) is the set $\mathsf{Sig}(\mathcal{O})$ ($\mathsf{Sig}(\alpha)$) of atomic concepts, atomic roles and individuals that occur in $\mathcal{O}$ (respectively in α).

The main reasoning task for ontologies is *entailment*: given an ontology $\mathcal{O}$ and an axiom α, check if $\mathcal{O}$ implies α. The logical entailment $\models$ is defined using the *usual Tarski-style set-theoretic semantics* for description logics as follows. An *interpretation* $\mathcal{I}$ is a pair $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is a non-empty set, called the *domain* of the interpretation, and $\cdot^{\mathcal{I}}$ is the *interpretation function* that assigns: to every $A \in \mathbf{C}$ a subset $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, to every $r \in \mathbf{R}$ a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and to every $a \in \mathbf{I}$ an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. Note that the sets $\mathbf{C}$, $\mathbf{R}$ and $\mathbf{I}$ are not defined by the interpretation $\mathcal{I}$ but assumed to be fixed for the ontology language (DL).

The interpretation function $\cdot^{\mathcal{I}}$ is extended to complex roles and concepts via DL-constructors as follows:

$$\begin{aligned}
(\top)^{\mathcal{I}} &= \Delta \\
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(\exists R.C)^{\mathcal{I}} &= \{x \in \Delta^{\mathcal{I}} \mid \exists y.\langle x,y\rangle \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} \\
(\neg C)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
(r^-)^{\mathcal{I}} &= \{\langle x,y\rangle \mid \langle y,x\rangle \in r^{\mathcal{I}}\} \\
(\geqslant n\, R)^{\mathcal{I}} &= \{\, x \in \Delta^{\mathcal{I}} \mid \sharp\{y \in \Delta^{\mathcal{I}} \mid \langle x,y\rangle \in R^{\mathcal{I}}\} \geq n\,\} \\
(\geqslant n\, R.C)^{\mathcal{I}} &= \{\, x \in \Delta^{\mathcal{I}} \mid \sharp\{y \in \Delta^{\mathcal{I}} \mid \langle x,y\rangle \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} \geq n\,\} \\
\{a\}^{\mathcal{I}} &= \{a^{\mathcal{I}}\}
\end{aligned}$$

The *satisfaction* relation $\mathcal{I} \models \alpha$ between an interpretation $\mathcal{I}$ and a DL axiom α (read as $\mathcal{I}$ *satisfies* α, or $\mathcal{I}$ is a *model* of α) is defined as follows:

$$\begin{array}{ll}
\mathcal{I} \models C_1 \sqsubseteq C_2 \text{ iff } C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}; & \mathcal{I} \models a : C \text{ iff } a^{\mathcal{I}} \in C^{\mathcal{I}}; \\
\mathcal{I} \models R_1 \sqsubseteq R_2 \text{ iff } R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}\,; & \mathcal{I} \models r(a,b) \text{ iff } \langle a^{\mathcal{I}}, b^{\mathcal{I}}\rangle \in r^{\mathcal{I}};
\end{array}$$

$$\mathcal{I} \models \mathsf{Trans}(r) \text{ iff } \forall x,y,z \in \Delta^{\mathcal{I}}[\,\langle x,y\rangle \in r^{\mathcal{I}} \wedge \langle y,z\rangle \in r^{\mathcal{I}} \Rightarrow \langle x,z\rangle \in r^{\mathcal{I}}\,];$$
$$\mathcal{I} \models \mathsf{Funct}(R) \text{ iff } \forall x,y,z \in \Delta^{\mathcal{I}}[\,\langle x,y\rangle \in R^{\mathcal{I}} \wedge \langle x,z\rangle \in R^{\mathcal{I}} \Rightarrow y = z\,];$$

An interpretation $\mathcal{I}$ is a *model* of an ontology $\mathcal{O}$ if $\mathcal{I}$ satisfies all axioms in $\mathcal{O}$. An ontology $\mathcal{O}$ *implies* an axiom α (written $\mathcal{O} \models \alpha$) if $\mathcal{I} \models \alpha$ for every model $\mathcal{I}$ of $\mathcal{O}$. Given a set $\mathbf{I}$ of interpretations, we say that an axiom α (an ontology $\mathcal{O}$) is *valid in* $\mathbf{I}$ if every interpretation $\mathcal{I} \in \mathbf{I}$ is a model of α (respectively $\mathcal{O}$). An axiom α is a *tautology* if it is valid in the set of all interpretations (or, equivalently, is implied by the empty ontology).

Let $\mathbf{S}_1, \mathbf{S}$ be signatures such that $\mathbf{S}_1 \subseteq \mathbf{S}$. The *restriction of an* $\mathbf{S}$*-interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ *to* $\mathbf{S}_1$ is an interpretation $\mathcal{I}|_{\mathbf{S}_1} = (\Delta^{\mathcal{I}_1}, \cdot^{\mathcal{I}_1})$ over $\mathbf{S}_1$ such that $\Delta^{\mathcal{I}_1} = \Delta^{\mathcal{I}}$ and $X^{\mathcal{I}_1} = X^{\mathcal{I}}$ for every $X \in \mathbf{S}_1$. An *expansion of an* $\mathbf{S}_1$*-interpretation* $\mathcal{I}_1$ *to* $\mathbf{S}$ is an $\mathbf{S}$-interpretation $\mathcal{I}$ such that $\mathcal{I}|_{\mathbf{S}_1} = \mathcal{I}_1$. A *trivial expansion of an* $\mathbf{S}_1$*-interpretation* $\mathcal{I}_1$ *to* $\mathbf{S}$ is an expansion of $\mathcal{I}_1$ to $\mathbf{S}$ such that $X^{\mathcal{I}} = \emptyset$ for every atomic concept and atomic role $X \in \mathbf{S} \setminus \mathbf{S}_1$.

6.3 Modules for Knowledge Reuse

Suppose that an ontology engineer wants to build an ontology about research projects. He defines two concepts Genetic_Disorder_Project and Cystic_Fibrosis_EUProject in his ontology $\mathcal{P}$. The first one describes projects about genetic disorders; the second one, European projects about cystic fibrosis, as given by the axioms P1 and P2 in Figure 6.1.

	Ontology of medical research projects $\mathcal{P}$:
P1	$\mathsf{Genetic\_Disorder\_Project} \equiv \mathsf{Project} \sqcap \exists \mathsf{has\_Focus}.\|\mathsf{Genetic\_Disorder}\|$
P2	$\mathsf{Cystic\_Fibrosis\_EUProject} \equiv \mathsf{EUProject} \sqcap \exists \mathsf{has\_Focus}\|\mathsf{Cystic\_Fibrosis}\|.$
P3	$\mathsf{EUProject} \sqsubseteq \mathsf{Project}$
	Ontology of medical terms $\mathcal{Q}$:
M1	$\|\mathsf{Cystic\_Fibrosis}\| \equiv \mathsf{Fibrosis} \sqcap \exists \mathsf{located\_In}.\mathsf{Pancreas} \sqcap \exists \mathsf{has\_Origin}.\mathsf{Genetic\_Origin}$
M2	$\mathsf{Genetic\_Fibrosis} \equiv \mathsf{Fibrosis} \sqcap \exists \mathsf{has\_Origin}.\mathsf{Genetic\_Origin}$
M3	$\mathsf{Fibrosis} \sqcap \exists \mathsf{located\_In}.\mathsf{Pancreas} \sqsubseteq \mathsf{Genetic\_Fibrosis}$
M4	$Genetic\_Fibrosis \sqsubseteq Genetic\_Disorder$
M5	$\mathsf{DEFBI\_Gene} \sqsubseteq \mathsf{Immuno\_Protein\_Gene} \sqcap \mathsf{associated\_WithCystic\_Fibrosis}$

Fig. 6.1. Reusing medical terminology in an ontology on research projects

The ontology engineer is an expert on research projects: he knows, for example, that a EUProject is a Project (axiom P3). He is unfamiliar, however, with most of the topics the projects cover and, in particular, with the terms Cystic_Fibrosis and Genetic_Disorder mentioned in P1 and P2. In this case, he decides to reuse the knowledge about these subjects from a well-established and widely-used medical ontology

The most straightforward way to reuse these concepts is to import the medical ontology. This may be, however, a large ontology, which deals with other matters in which the ontology engineer is not interested, such as genes, anatomy, surgical techniques, etc. Ideally, one would like to extract a (hopefully small) fragment of the medical ontology—a *module*—that describes in detail the concepts we are reusing in our ontology. Intuitively, importing the module $\mathcal{Q}_1$ into $\mathcal{P}$ instead of the full ontology $\mathcal{Q}$ should have no impact on the modeling of the ontology $\mathcal{P}$.

Suppose that the concepts Cystic_Fibrosis and Genetic_Disorder are described in an ontology $\mathcal{Q}$ containing axioms M1-M5 in Figure 6.1. If we include in the module

$\mathcal{Q}_1$ just the axioms that mention either Cystic_Fibrosis or Genetic_Disorder, namely M1, M4 and M5, we lose the following dependency:

$$\mathsf{Cystic\_Fibrosis} \sqsubseteq \mathsf{Genetic\_Disorder} \tag{6.1}$$

The dependencies Cystic_Fibrosis $\sqsubseteq$ Genetic_Fibrosis $\sqsubseteq$ Genetic_Disorder follow from axioms M1-M5, but not from M1, M4, M5, since the dependency Cystic_Fibrosis $\sqsubseteq$ Genetic_Fibrosis does not hold after removing M2 and M3. The dependency (6.1), however, is crucial for our ontology $\mathcal{P}$ as it (together with axiom P3) implies the following axiom:

$$\mathsf{Cystic\_Fibrosis\_EUProject} \sqsubseteq \mathsf{Genetic\_Disorder\_Project} \tag{6.2}$$

This means, in particular, that all the projects annotated with the concept name Cystic_Fibrosis_EUProject must be included in the answer for a query on the concept name Genetic_Disorder_Project. Consequently, importing a part of $\mathcal{Q}$ containing only axioms that mention the terms used in $\mathcal{P}$ instead of $\mathcal{Q}$ results in an underspecified ontology. We stress that the ontology engineer might be unaware of dependency (6.2), even though it concerns the concepts of his primary scope.

The example above suggests that the central requirement for a module $\mathcal{Q}_1 \subseteq \mathcal{Q}$ to be reused in our ontology $\mathcal{P}$ is that $\mathcal{P} \cup \mathcal{Q}_1$ should yield the *same* logical consequences in the vocabulary of $\mathcal{P}$ as $\mathcal{P} \cup \mathcal{Q}$ does. Note that, as seen in the example, this requirement does not force us to include in $\mathcal{Q}_1$ all the axioms in $\mathcal{Q}$ that mention the vocabulary to be reused, nor does it imply that the axioms in $\mathcal{Q}$ that do not mention this vocabulary should be omitted.

Based on the discussion above, we formalize our first notion of a *module* as follows:

Definition 1 (Module). *Let* L *be a description logic,* $\mathcal{Q}_1 \subseteq \mathcal{Q}$ *be two ontologies expressed in* L *and let* **S** *be a signature. We say that* $\mathcal{Q}_1$ *is an* **S**-module in $\mathcal{Q}$ *w.r.t.* L*, if for every ontology* $\mathcal{P}$ *and every axiom* α *expressed in* L *with* $\mathsf{Sig}(\mathcal{P} \cup \{\alpha\}) \cap \mathsf{Sig}(\mathcal{Q}) \subseteq$ **S***, we have* $\mathcal{P} \cup \mathcal{Q} \models \alpha$ *iff* $\mathcal{P} \cup \mathcal{Q}_1 \models \alpha$*.* ◇

In Definition 1 the signature **S** acts as the *interface* signature between $\mathcal{P}$ and $\mathcal{Q}$ in the sense that it contains the symbols that $\mathcal{P}$ and α may share with $\mathcal{Q}$. It is also important to realize that there are two free parameters in Definition 1, namely the ontology $\mathcal{P}$ and the axiom α. Both $\mathcal{P}$ and α are formulated in some ontology language L, which might not necessarily be OWL DL.

Fixing the language L in which $\mathcal{P}$ and α can be expressed is essential in Definition 1 since it may well be the case that $\mathcal{Q}_1$ is a module in $\mathcal{Q}$ w.r.t. a language L_1, but not w.r.t. L_2. Fixing L, however, is not always reasonable. If $\mathcal{Q}_1$ is an **S**-module in $\mathcal{Q}$, it should always be possible to replace $\mathcal{Q}$ with $\mathcal{Q}_1$ regardless of the particular language in which $\mathcal{P}$ and α are expressed. In fact, we may extend our ontology $\mathcal{P}$ with a set of Horn rules, or extend our query language to support arbitrary conjunctive queries. In any case, extending the ontology language for $\mathcal{P}$ and the query language for α should not prevent $\mathcal{Q}_1$ from being a module in $\mathcal{Q}$.

It is therefore convenient to formulate a more general notion of a module which abstracts from the particular language under consideration; that is, we say that $\mathcal{Q}_1$

is an $\mathbf{S}$-module in $\mathcal{Q}$ iff it is an $\mathbf{S}$-module in $\mathcal{Q}$, according to Definition 1 for *every* language L with Tarski-style set-theoretic semantics. The modules we obtain in this paper will be modules in precisely this stronger sense.

According to Definition 1, there may be a large number of modules for a given input ontology and signature. In many applications one is usually not interested in extracting arbitrary modules from a reused ontology, but in extracting modules that are easy to process afterwards. Ideally, the extracted modules should be as small as possible. Hence, it is reasonable to consider the problem of extracting *minimal modules*—that is, modules that contain no other module as a subset In our example from Figure 6.1, there are two minimal $\mathbf{S}$-modules $\mathcal{Q}_1 = \{\mathcal{M}1, \mathcal{M}2, \mathcal{M}4\}$ and $\mathcal{Q}_2 = \{\mathcal{M}1, \mathcal{M}3, \mathcal{M}4\}$: if we remove any axiom from them, the dependency (6.1) will no longer hold.

As seen above, minimal modules are not necessarily unique. While in some cases it is reasonable to extract all minimal modules, in others it may suffice to extract just one. Thus, given $\mathcal{Q}$ and $\mathbf{S}$, the following tasks are of interest:

$$\begin{array}{l} \text{T1. compute } \textit{all} \text{ minimal } \mathbf{S}\text{-modules in } \mathcal{Q} \\ \text{T2. compute } \textit{some} \text{ minimal } \mathbf{S}\text{-module in } \mathcal{Q} \end{array} \tag{6.3}$$

Axioms that do not occur in a minimal module of $\mathcal{Q}$ are not essential for $\mathcal{P}$ in the sense that they do not need to be imported into $\mathcal{P}$. This is not true for the axioms that occur in minimal modules of $\mathcal{Q}$. These arguments motivate the following notion:

Definition 2 (Essential Axiom). *Given a signature* $\mathbf{S}$ *and an ontology* $\mathcal{Q}$*, we say that an axiom* $\alpha \in \mathcal{Q}$ *is* $\mathbf{S}$-essential in $\mathcal{Q}$ *w.r.t.* L *if* α *belongs to some minimal* $\mathbf{S}$*-module in* $\mathcal{Q}$ *w.r.t.* L. $\Diamond$

In our example, the axioms $\mathcal{M}1 - \mathcal{M}4$ from $\mathcal{Q}$ are essential for the signature $\mathbf{S} = \{\mathsf{CysticFibrosis}, \mathsf{Genetic\_Disorder}\}$, and the axiom $\mathcal{M}5$ is not essential. In certain situations one might be interested in computing the set of (non)essential axioms of an ontology, which can be done by computing the union of all minimal modules. Hence, the following task may also be of interest:

$$\begin{array}{l} \text{T3. compute } \textit{the union} \text{ of all minimal } \mathbf{S}\text{-modules in } \mathcal{Q}, \\ \quad\;\; \text{which is the set of all } \mathbf{S}\text{-essential axioms in } \mathcal{Q} \end{array} \tag{6.4}$$

Note that computing the union of minimal modules might be easier than computing all the minimal modules since one does not need to identify which axiom belongs to which minimal module.

6.4 Computational Properties of Module Extraction

In this section, we study the decidability/computability of the tasks described in Section 6.3. We start by investigating the relationships between Tasks T1,T2 and T3; our main result is that tasks T1 and T2 are inter-reducible whereas T3 is "easier" than both T1 and T2. Next, we establish the relationship between our notion of module and the notion of a *conservative extension*, whose complexity/decidability for Description Logics has been recently established. Finally, we show that deciding whether an axiom is essential in an ontology for a given signature is an undecidable problem for the logic $\mathcal{ALCO}$. As a consequence, tasks T1-T3 are proved algorithmically unsolvable for $\mathcal{ALCO}$.

6.4.1 Reductions between Tasks

Before we formalize we establish the reductions between tasks T1-T3, we prove in the following proposition some important properties of modules that we will exploit along this section.

Proposition 1 (Properties of Modules)
Let $\mathcal{Q}_1 \subseteq \mathcal{Q}_2 \subseteq \mathcal{Q}_3$ be three $\mathcal{SHOIQ}$ ontologies and $\mathbf{S}$ *be a signature. Then:*

1. If $\mathcal{Q}_1$ is an $\mathbf{S}$*-module in $\mathcal{Q}_2$ and $\mathcal{Q}_2$ is an* $\mathbf{S}$*-module in $\mathcal{Q}_3$ then*
$\mathcal{Q}_1$ is an $\mathbf{S}$*-module in $\mathcal{Q}_3$ (transitivity)*
2. If $\mathcal{Q}_1$ is an $\mathbf{S}$*-module in $\mathcal{Q}_3$ then*
(a) $\mathcal{Q}_1$ is an $\mathbf{S}$*-module in $\mathcal{Q}_2$ and (b) $\mathcal{Q}_2$ is an* $\mathbf{S}$*-module in $\mathcal{Q}_3$ (convexity)*

Proof

1. Suppose that $\mathcal{Q}_1$ is an $\mathbf{S}$-module in $\mathcal{Q}_2$ and $\mathcal{Q}_2$ is an $\mathbf{S}$-module in $\mathcal{Q}_3$. In order to prove that $\mathcal{Q}_1$ is an $\mathbf{S}$-module in $\mathcal{Q}_3$ according to Definition 1, take any ontology $\mathcal{P}$ and an axiom α such that $\mathsf{Sig}(\mathcal{P} \cup \{\alpha\}) \cap \mathsf{Sig}(\mathcal{Q}_3) \subseteq \mathbf{S}$ and $\mathcal{P} \cup \mathcal{Q}_3 \models \alpha$. We demonstrate that $\mathcal{P} \cup \mathcal{Q}_1 \models \alpha$ ($\star$):

Since $\mathcal{Q}_2$ is an $\mathbf{S}$-module in $\mathcal{Q}_3$, $\mathsf{Sig}(\mathcal{P} \cup \{\alpha\}) \cap \mathsf{Sig}(\mathcal{Q}_3) \subseteq \mathbf{S}$ and $\mathcal{P} \cup \mathcal{Q}_3 \models \alpha$, we have $\mathcal{P} \cup \mathcal{Q}_2 \models \alpha$. Since $\mathcal{Q}_1$ is an $\mathbf{S}$-module in $\mathcal{Q}_2$, $\mathsf{Sig}(\mathcal{P} \cup \{\alpha\}) \cap \mathsf{Sig}(\mathcal{Q}_2) \subseteq \mathsf{Sig}(\mathcal{P} \cup \{\alpha\}) \cap \mathsf{Sig}(\mathcal{Q}_3) \subseteq \mathbf{S}$, and $\mathcal{P} \cup \mathcal{Q}_2 \models \alpha$, we have $\mathcal{P} \cup \mathcal{Q}_1 \models \alpha$ ($\star$).

2.(a) Suppose that $\mathcal{Q}_1$ is an $\mathbf{S}$-module in $\mathcal{Q}_3$. In order to prove that $\mathcal{Q}_1$ is an $\mathbf{S}$-module in $\mathcal{Q}_2$, consider any ontology $\mathcal{P}$ and an axiom α such that $\mathsf{Sig}(\mathcal{P} \cup \{\alpha\}) \cap \mathsf{Sig}(\mathcal{Q}_2) \subseteq \mathbf{S}$ and $\mathcal{P} \cup \mathcal{Q}_2 \models \alpha$. We demonstrate that $\mathcal{P} \cup \mathcal{Q}_1 \models \alpha$ ($\sharp$):

Without loss of generality, we can assume that $\mathsf{Sig}(\mathcal{P} \cup \{\alpha\}) \cap \mathsf{Sig}(\mathcal{Q}_3) \subseteq \mathbf{S}$, since the symbols that are in $\mathsf{Sig}(\mathcal{P} \cup \{\alpha\})$ but not in $\mathsf{Sig}(\mathcal{Q}_2)$ could be renamed so that they are not contained in $\mathsf{Sig}(\mathcal{Q}_3)$. Since $\mathcal{Q}_1$ is an $\mathbf{S}$-module in $\mathcal{Q}_3$ and $\mathcal{P} \cup \mathcal{Q}_3 \models \mathcal{P} \cup \mathcal{Q}_2 \models \alpha$, we have $\mathcal{P} \cup \mathcal{Q}_1 \models \alpha$ ($\sharp$).

2.(b) Suppose that $\mathcal{Q}_1$ is an $\mathbf{S}$-module in $\mathcal{Q}_3$. In order to prove that $\mathcal{Q}_2$ is an $\mathbf{S}$-module in $\mathcal{Q}_3$, consider any ontology $\mathcal{P}$ and an axiom α such that $\mathsf{Sig}(\mathcal{P} \cup \{\alpha\}) \cap \mathsf{Sig}(\mathcal{Q}_3) \subseteq \mathbf{S}$ and $\mathcal{P} \cup \mathcal{Q}_3 \models \alpha$. We demonstrate that $\mathcal{P} \cup \mathcal{Q}_2 \models \alpha$ ($\dagger$):

Since $\mathcal{Q}_1$ is an $\mathbf{S}$-module in $\mathcal{Q}_3$, $\mathsf{Sig}(\mathcal{P} \cup \{\alpha\}) \cap \mathsf{Sig}(\mathcal{Q}_3) \subseteq \mathbf{S}$, and $\mathcal{P} \cup \mathcal{Q}_3 \models \alpha$, we have $\mathcal{P} \cup \mathcal{Q}_1 \models \alpha$. Since $\mathcal{Q}_1 \subseteq \mathcal{Q}_2$, we have $\mathcal{P} \cup \mathcal{Q}_2 \models \alpha$ ($\dagger$). $\diamond$

Intuitively, Part 2(a) of Proposition 1 claims that every superset of an $\mathbf{S}$-module of this ontology is also an $\mathbf{S}$-module of the ontology. This means, in particular, that it is sufficient to compute only the minimal modules of an ontology in order to have a complete information about all the modules.

Intuitively, task T2 should be simpler than T1. That is, any procedure which solves the task T1, also provides a solution for task T2. Surprisingly, the converse of this property holds as well: any procedure for T2 can be turned into a procedure for T1. The following lemma is the key property underlying this reduction:

Lemma 1 (A Criterium for Minimal Modules)
Let $\mathcal{Q}$ be an ontology and $\mathbf{S}$ *be a signature. Let $\mathcal{M}$ be the set of all subsets $\mathcal{Q}_2$ of $\mathcal{Q}$ such that $\mathcal{Q}_2$ is a minimal (and hence is the only)* $\mathbf{S}$*-module in $\mathcal{Q}_2$.*

Then $\mathcal{Q}_1$ is a minimal $\mathbf{S}$*-module in $\mathcal{Q}$ iff (i) $\mathcal{Q}_1 \in \mathcal{M}$, and (ii) there is no $\mathcal{Q}_2 \in \mathcal{M}$ such that $\mathcal{Q}_1 \subsetneq \mathcal{Q}_2$.*

Proof

$(\Rightarrow)$ Suppose $\mathcal{Q}_1$ is a minimal $\mathbf{S}$-module in $\mathcal{Q}$. We need to show that properties (i) and (ii) above hold for $\mathcal{Q}_1$.

(i) Suppose, to the contrary, that the property (i) does not hold for $\mathcal{Q}_1$, i.e. $\mathcal{Q}_1$ is not a minimal module in $\mathcal{Q}_1$. Then there exists a $\mathcal{Q}_2 \subsetneq \mathcal{Q}_1 \subseteq \mathcal{Q}$ such that $\mathcal{Q}_2$ is an $\mathbf{S}$-module in $\mathcal{Q}_1$. Since $\mathcal{Q}_1$ is an $\mathbf{S}$-module in $\mathcal{Q}$, By the part 1 of Proposition 1 (transitivity), $\mathcal{Q}_2$ is an $\mathbf{S}$-module in $\mathcal{Q}$. Hence $\mathcal{Q}_1$ is not a minimal module in $\mathcal{Q}$ contrary to what has been assumed.

(ii) Suppose, to the contrary, that the property (ii) does not hold for $\mathcal{Q}_1$, that is, there exists $\mathcal{Q}_2 \in \mathcal{M}$ such that $\mathcal{Q}_1 \subsetneq \mathcal{Q}_2 \subseteq \mathcal{Q}$. Since $\mathcal{Q}_1$ is an $\mathbf{S}$-module in $\mathcal{Q}$, by the part 2(a) of Proposition 1, $\mathcal{Q}_1$ is an $\mathbf{S}$-module in $\mathcal{Q}_2$. Hence $\mathcal{Q}_2 \notin \mathcal{M}$ by the definition of $\mathcal{M}$ (since $\mathcal{Q}_2$ is not a minimal $\mathbf{S}$-module in $\mathcal{Q}_2$), which yields a contradiction.

$(\Leftarrow)$ Assume that conditions (i) and (ii) above hold for $\mathcal{Q}_1$, but $\mathcal{Q}_1$ is not a minimal $\mathbf{S}$-module in $\mathcal{Q}$. There are two cases possible: (a) $\mathcal{Q}_1$ is not an $\mathbf{S}$-module in $\mathcal{Q}$, and (b) $\mathcal{Q}_1$ is an $\mathbf{S}$-module in $\mathcal{Q}$, but not a minimal $\mathbf{S}$-module.

In the case (a), there has to be a minimal $\mathbf{S}$-module $\mathcal{Q}_2$ in $\mathcal{Q}$ such that $\mathcal{Q}_1 \subsetneq \mathcal{Q}_2 \subseteq \mathcal{Q}$. By the direction $(\Rightarrow)$ of the lemma applied to $\mathcal{Q}_2$, we have $\mathcal{Q}_2 \in \mathcal{M}$. But this contradicts the condition (ii), since $\mathcal{Q}_1 \in \mathcal{M}$ and $\mathcal{Q}_1 \subsetneq \mathcal{Q}_2$.

In the case (b), there is a minimal $\mathbf{S}$-module $\mathcal{Q}_2$ in $\mathcal{Q}$ such that $\mathcal{Q}_2 \subsetneq \mathcal{Q}_1 \subseteq \mathcal{Q}$. By the property 2.(a) of Proposition 1, $\mathcal{Q}_2$ is an $\mathbf{S}$-module in $\mathcal{Q}_1$, which contradicts the condition (i) since $\mathcal{Q}_1$ is not a minimal $\mathbf{S}$-module in $\mathcal{Q}_1$. ◇

We use this property to show that tasks T1 and T2 are indeed inter-reducible:

Proposition 2. *Tasks T1 and T2 from* (6.3) *are inter-reducible.*

Proof. As it has been already pointed out, using a procedure for task T1 one can obtain a procedure for task T2 by just returning any of the computed minimal $\mathbf{S}$-modules in $\mathcal{Q}$.

Now suppose we have a procedure **P2** for task T2, namely, that given a signature $\mathbf{S}$ and an ontology $\mathcal{Q}$ returns some minimal $\mathbf{S}$-module $\mathcal{Q}_1$ in $\mathcal{Q}$. We construct a procedure **P1** that returns all minimal $\mathbf{S}$-modules, which is based on the criterium for minimal $\mathbf{S}$-modules formulated in Lemma 1. Note that procedure **P2** satisfies the following property:

Given $\mathbf{S}$ *and $\mathcal{Q}_2$, the procedure* ***P2*** *for T2 returns $\mathcal{Q}_2$ if and only if $\mathcal{Q}_2$ is the only minimal* $\mathbf{S}$*-module in $\mathcal{Q}_2$.* (6.5)

Procedure **P1** should work as follows. Given **S** and $\mathcal{Q}$, **P1** first computes the set $\mathcal{M}$ of subsets $\mathcal{Q}_2$ in $\mathcal{Q}$ such that $\mathcal{Q}_2$ is the only **S**-module in $\mathcal{Q}_2$ using property (6.5) of procedure **P2**. More precisely, in order to compute $\mathcal{M}$, we enumerate all the subsets of $\mathcal{Q}$ and select those subsets $\mathcal{Q}_2$ for which **P2** returns $\mathcal{Q}_2$. Next, **P1** returns those sets from $\mathcal{M}$ that are contained in no other set from $\mathcal{M}$. By Lemma 1, **P1** returns exactly all minimal **S**-modules in $\mathcal{Q}$. $\diamond$

Obviously, task T3 is at least not harder then task T1:

Proposition 3. *Tasks T1 and T2 are reducible to task T3; that is, any procedure for T1 or T2 can be used for solving T3.*

It is not clear, however, whether the procedure for T3 can be used to obtain a procedure for T1. Nevertheless, as we will demonstrate Section 6.4.2, this issue is not relevant since all of the tasks formulated above are algorithmically unsolvable for OWL DL.

6.4.2 Modules and Conservative Extensions

The notion of a module is closely related to the notion of a conservative extension which has been used to characterize formal requirements in ontology integration tasks [7, 5, 4, 12]. In the literature we can find at least two different notions of conservative extensions in the context of ontologies [12]:

Definition 3 (Conservative Extensions)
Let $\mathcal{Q}_1 \subseteq \mathcal{Q}$ *be two ontologies,* **S** *a signature and* L *a logic. We say that* $\mathcal{Q}$ *is a* deductive **S**-conservative extension *of* $\mathcal{Q}_1$ *w.r.t.* L*, if for every axiom* α *over* L *with* $\mathsf{Sig}(\alpha) \subseteq \mathbf{S}$*, we have* $\mathcal{Q} \models \alpha$ *iff* $\mathcal{Q}_1 \models \alpha$*. We say that* $\mathcal{Q}$ *is a* model **S**-conservative extension *of* $\mathcal{Q}_1$ *if, for every model* $\mathcal{I}_1$ *of* $\mathcal{Q}_1$*, there exists a model* $\mathcal{I}$ *of* $\mathcal{Q}$ *such that* $\mathcal{I}|_{\mathbf{S}} = \mathcal{I}_1|_{\mathbf{S}}$. $\diamond$

Intuitively, an ontology $\mathcal{Q}$ is a deductive conservative extension of an ontology $\mathcal{Q}_1 \subseteq \mathcal{Q}$ for a signature **S** iff every logical consequence α of $\mathcal{Q}$ constructed using only symbols from **S** is already a consequence of $\mathcal{Q}_1$; that is, the additional axioms in $\mathcal{Q}$ do not add new logical consequences over the vocabulary **S**. Analogously to modules, the notion of a deductive conservative extension depends on the ontology language L in which $\mathcal{Q}$ and α are expressed.

In contrast, model conservative extensions are not defined in terms of logical entailment, but using the models directly. Intuitively, an ontology $\mathcal{Q}$ is a model conservative extension of $\mathcal{Q}_1 \subseteq \mathcal{Q}$ if every model of $\mathcal{Q}_1$ can be expanded to a model of $\mathcal{Q}$ by interpreting new symbols and leaving the interpretations of the old symbols unchanged.

The notion of model conservative extension is strictly stronger than the deductive one [12] since it does not depend on expressivity of the ontology language. That is, if $\mathcal{Q}$ is a model **S**-conservative extension of $\mathcal{Q}_1$, it is also a deductive **S**-conservative extension of $\mathcal{Q}_1$, but not necessarily vice versa.

Example 1. Let $\mathcal{Q}$ be the ontology consisting of axioms $\mathcal{M}1 - \mathcal{M}5$ in Figure 6.1. Let $\mathbf{S} = \{\mathsf{Cystic\_Fibrosis}, \mathsf{Genetic\_Disorder}\}$ and $\mathcal{Q}_1 = \{\mathcal{M}1, \ldots, \mathcal{M}4\}$. We show that $\mathcal{Q}$ is a model $\mathbf{S}$-conservative extension of $\mathcal{Q}_1$ and, hence, also a deductive conservative extension of $\mathcal{Q}_1$.

Let $\mathcal{I}_1$ be an arbitrary model of $\mathcal{Q}_1$. We demonstrate that we can always construct a model $\mathcal{I}$ of $\mathcal{Q}$ which interprets the symbols from $\mathbf{S}$ in the same way as $\mathcal{I}_1$ does, i.e. $\mathcal{I}|_\mathbf{S} = \mathcal{I}_1|_\mathbf{S}$.

Let $\mathcal{I}$ be as $\mathcal{I}_1$ except for the interpretation of the atomic concepts $\mathsf{DEFBI\_Gene}$ and $\mathsf{Immuno\_Protein\_Gene}$, and the atomic role $\mathsf{associatedWith}$, all of which we interpret in $\mathcal{I}$ as the empty set. Note that these atomic concepts and this atomic role do not occur in $\mathcal{Q}_1$. Hence, $\mathcal{I}$ interprets the concepts in $\mathcal{Q}_1$ exactly like $\mathcal{I}_1$, and so $\mathcal{I}$ is a model of $\mathcal{Q}_1$. Furthermore, $\mathcal{I}$ is a model of M5 since the concepts on the left-hand-side and the right-hand-side of this axiom are both interpreted as the empty set. Thus, $\mathcal{Q}$ is a model $\mathbf{S}$-conservative extension of $\mathcal{Q}_1$.

In fact, it was sufficient to take any expansion $\mathcal{I}$ of $\mathcal{I}_1$ in which $\mathsf{DEFBI\_Gene}$ is interpreted as the empty set. Hence $\mathcal{Q}$ is a model $\mathbf{S}$-conservative extension of $\mathcal{Q}_1$ for every $\mathbf{S}$ that does not contain $\mathsf{DEFBI\_Gene}$ since $\mathcal{M}5$ is satisfied in every interpretation where this concept is interpreted as the empty set.

Now, if we remove M2 and M3 from $\mathcal{Q}_1$, then $\mathcal{Q}$ is no longer a model $\mathbf{S}$-conservative extension of $\mathcal{Q}_1$ for $\mathbf{S} = \{\mathsf{Cystic\_Fibrosis}, \mathsf{Genetic\_Disorder}\}$. Indeed, it is possible to find an interpretation $\mathcal{I}_1$ of the remaining axioms M1 and M4 from $\mathcal{O}_1$, in which $\mathsf{Genetic\_Disorder}$ is interpreted as the empty set, but $\mathsf{Cystic\_Fibrosis}$ is not. For example, consider an interpretation $\mathcal{I}_1 = (\{a\}, \cdot^{\mathcal{I}_1})$ with:

$$\begin{array}{l}\mathsf{Cystic\_Fibrosis}^{\mathcal{I}_1} = \mathsf{Fibrosis}^{\mathcal{I}_1} = \mathsf{Pancreas}^{\mathcal{I}_1} = \mathsf{Genetic\_Origin}^{\mathcal{I}_1} = \{a\};\\ \mathsf{located\_In}^{\mathcal{I}_1} = \mathsf{has\_Origin}^{\mathcal{I}_1} = \{(a,a)\}; \quad \text{and}\\ \mathsf{Genetic\_Fibrosis}^{\mathcal{I}_1} = \mathsf{Genetic\_Disorder}^{\mathcal{I}_1} = \emptyset.\end{array}$$

We cans see that $\mathcal{I}_1$ is a model of M1 and M4, but there is no model $\mathcal{I}$ of $\mathcal{Q}$ such that $\mathcal{I}|_\mathbf{S} = \mathcal{I}_1|_\mathbf{S}$. Indeed, for every model $\mathcal{I}$ of $\mathcal{Q}$, we must have $\mathcal{I} \models \alpha :=$ $(\mathsf{Cystic\_Fibrosis} \sqsubseteq \mathsf{Genetic\_Disorder})$ because $\mathcal{Q} \models \alpha$. However, this would imply also that $\mathcal{I}_1 \models \alpha$, since $\mathcal{I}|_\mathbf{S} = \mathcal{I}_1|_\mathbf{S}$, but this does not hold for $\mathcal{I}_1$ defined above. ◇

Although Definition 1 is close to the notion of deductive conservative extension, there are two important differences. First, in the definition of deductive conservative extension, the logical consequences are considered only w.r.t. the ontologies $\mathcal{Q}$ and $\mathcal{Q}_1$ of interest whereas, in our definition of module, all the possible ontologies $\mathcal{P}$ in which the module can be used are taken into account. Second, in the definition of deductive conservative extension, the signature of α is required to be a subset of $\mathbf{S}$ whereas, in our definition of module, only the common part of $\{\alpha\} \cup \mathcal{P}$ and $\mathcal{Q}$ is required to be a subset of $\mathbf{S}$. Despite these differences, the two notions of conservative extensions are related to our notion of module:

Proposition 4 (Modules vs. Conservative Extensions)
Let $\mathcal{Q}_1 \subseteq \mathcal{Q}$ be two ontologies. Then:

1. *If $\mathcal{Q}_1$ is an* $\mathbf{S}$*-module in $\mathcal{Q}$ w.r.t.* L *then $\mathcal{Q}$ is a deductive* $\mathbf{S}$*-conservative extension of $\mathcal{Q}_1$ w.r.t.* L*;*

2. *If $\mathcal{Q}$ is a model* $\mathbf{S}$*-conservative extension of $\mathcal{Q}_1$ then $\mathcal{Q}_1$ is an* $\mathbf{S}$*-module in $\mathcal{Q}$ for every ontology language* L *with Tarski-style set-theoretic semantics.*

Proof

1. Let α be an axiom with $\mathsf{Sig}(\alpha) \in \mathbf{S}$ such that $\mathcal{Q} \models \alpha$. We have to show that $\mathcal{Q}_1 \models \alpha$ $(\star)$. Take $\mathcal{P} := \emptyset$ (the empty ontology). Since $\mathcal{Q}_1$ is a module in $\mathcal{Q}$, $\mathsf{Sig}(\mathcal{P} \cup \{\alpha\}) \cap \mathsf{Sig}(\mathcal{Q}) \subseteq \mathbf{S}$, and $\mathcal{P} \cup \mathcal{Q} = \mathcal{Q} \models \alpha$, by Definition 1, we have $\mathcal{Q}_1 = \mathcal{P} \cup \mathcal{Q}_1 \models \alpha$.

2. Assume that $\mathcal{Q}$ is a model $\mathbf{S}$-conservative extension of $\mathcal{Q}_1$, but $\mathcal{Q}_1$ is not an $\mathbf{S}$-module in $\mathcal{Q}$ w.r.t. some logic L. According to Definition 1, this means that there exists an ontology $\mathcal{P}$ and an axiom α over L with $\mathsf{Sig}(\mathcal{P} \cup \{\alpha\}) \cap \mathsf{Sig}(\mathcal{Q}) \subseteq \mathbf{S}$, such that $\mathcal{P} \cup \mathcal{Q} \models \alpha$ but $\mathcal{P} \cup \mathcal{Q}_1 \not\models \alpha$. The last implies that for some interpretation $\mathcal{I}_1$, we have $\mathcal{I}_1 \models \mathcal{P} \cup \mathcal{Q}_1$, but $\mathcal{I}_1 \not\models \alpha$. Let $\mathcal{I}'_1 := \mathcal{I}_1|_{\mathbf{S} \cup \mathsf{Sig}(\mathcal{Q})}$. Obviously, $\mathcal{I}'_1 \models \mathcal{Q}_1$. By Definition 3, since $\mathcal{Q}$ is a model $\mathbf{S}$-conservative extension of $\mathcal{Q}_1$, there exists an interpretation $\mathcal{I}'$ such that $\mathcal{I}' \models \mathcal{Q}$ and $\mathcal{I}'|_{\mathbf{S}} = \mathcal{I}'_1|_{\mathbf{S}}$. Let $\mathcal{I}$ be the expansion of $\mathcal{I}'|_{\mathbf{S} \cup \mathsf{Sig}(\mathcal{Q})}$ to $\mathsf{Sig}(\mathcal{P} \cup \{\alpha\})$ by setting $X^{\mathcal{I}} := X^{\mathcal{I}_1}$ for every $X \in \mathsf{Sig}(\mathcal{P} \cup \{\alpha\}) \setminus \mathbf{S}$. Note that we also have $\mathcal{I}|_{\mathbf{S}} = \mathcal{I}'|_{\mathbf{S}} = \mathcal{I}'_1|_{\mathbf{S}} = \mathcal{I}_1|_{\mathbf{S}}$, hence $\mathcal{I}|_{\mathsf{Sig}(\mathcal{P} \cup \{\alpha\})} = \mathcal{I}_1|_{\mathsf{Sig}(\mathcal{P} \cup \{\alpha\})}$, and so $\mathcal{I} \models \mathcal{P}$ and $\mathcal{I} \not\models \alpha$. Since $\mathcal{I}|_{\mathbf{S} \cup \mathsf{Sig}(\mathcal{Q})} = \mathcal{I}'|_{\mathbf{S} \cup \mathsf{Sig}(\mathcal{Q})}$ and $\mathcal{I}' \models \mathcal{Q}$, we have $\mathcal{I} \models \mathcal{Q}$, which yields a contradiction. $\diamond$

Proposition 4 shows that our notion of module stays "in between" the two notions of conservative extensions. In particular, by applying Property 2 in Proposition 4 to Example 1, we can show that the axioms M1-M4 in Figure 6.1 constitute a module in the ontology $\mathcal{Q}$, consisting of M1-M5. The converse of Property 1 in Proposition 4, however, does not hold in general:

Example 2. Let $\mathcal{Q}_1 = \{\}$, $\mathcal{Q} = \{\top \sqsubseteq \exists R.A\}$ and $\mathbf{S} = \{A\}$. The ontology $\mathcal{Q}$ is a deductive $\mathbf{S}$-conservative extension of $\mathcal{Q}_1$ w.r.t. $\mathcal{ALC}$. Indeed, every $\mathcal{ALC}$-axiom $\alpha = (C_1 \sqsubseteq C_2)$ over $\mathbf{S} = \{A\}$, is equivalent in $\mathcal{ALC}$ to either $\top \sqsubseteq \top$, $\top \sqsubseteq \bot$, $\top \sqsubseteq A$ or $A \sqsubseteq \bot$, which are indistinguishable by $\mathcal{Q}_1$ and $\mathcal{Q}$—that is, the axiom is implied by $\mathcal{Q}_1$ iff it is implied by $\mathcal{Q}$. $\mathcal{Q}_1$, however, is not an $\mathbf{S}$-module in $\mathcal{Q}$. Consider an $\mathcal{ALC}$-ontology $\mathcal{P} = \{A \sqsubseteq \bot\}$, which is constructed over $\mathbf{S}$. We can see that $\mathcal{P} \cup \mathcal{Q} \models \top \sqsubseteq \bot$, but $\mathcal{P} \cup \mathcal{Q}_1 \not\models \top \sqsubseteq \bot$. $\diamond$

Note that the construction in Example 2 also shows that the notion of deductive conservative extension is strictly weaker than the notion of model conservative extension (as shown in [12]): $\mathcal{Q}$ is a deductive conservative extension of $\mathcal{Q}_1$ but, according to Property 2 in Proposition 4, it is not a model conservative extension.

6.4.3 Undecidability Results

Given the relationships between our definition of module and conservative extensions, it is worth examining the computational complexity of the associated problems. The problem of deciding whether $\mathcal{Q}$ is an $\mathbf{S}$-conservative extension of $\mathcal{Q}_1$ has been studied in [12], where it is proved to be 2-EXPTIME complete for $\mathcal{ALCIQ}$

(roughly OWL-Lite) and undecidable for OWL DL. For model conservative extensions, the problem is highly undecidable (non recursively enumerable), even for $\mathcal{ALC}$ [12].

The decidability result from [12] for deductive conservative extensions, however, does not transfer to our problem since an ontology $\mathcal{Q}$ may well be an $\mathbf{S}$-deductive conservative extension of $\mathcal{Q}_1$, but still $\mathcal{Q}_1$ might not be an $\mathbf{S}$-module in $\mathcal{Q}$. In fact, we show that our problem is already undecidable for $\mathcal{ALCO}$ ontologies:

Theorem 1 (Undecidability for Essential Axioms)
Given a signature $\mathbf{S}$*, an* $\mathcal{ALC}$*-ontology* $\mathcal{Q}$ *and an axiom* $\alpha \in \mathcal{Q}$*, it is undecidable whether* α *is* $\mathbf{S}$*-essential in* $\mathcal{Q}$ *w.r.t.* $\mathrm{L} = \mathcal{ALCO}$*.*

Proof. The proof is a variation of the construction for undecidability of deciding deductive conservative extensions in $\mathcal{ALCQIO}$ given [12], based on a reduction from a domino tiling problem.

A domino system is a triple $D = (T, H, V)$ where T is a finite set of *tiles* and $H, V \subseteq T \times T$ are *horizontal* and *vertical matching relations*. A *solution* for a domino system D is a mapping $t_{(\cdot,\cdot)}$ that assigns to every pair of integers $i, j \geq 1$ an element $t_{i,j} \in T$, such that $(t_{i,j}, t_{i,j+1}) \in V$ and $(t_{i,j}, t_{i+1,j}) \in H$. A *periodic solution* for a domino system D is a solution $t_{i,j}$ for which there exist integers $m \geq 1$, $n \geq 1$ called *periods* such that $t_{i+m,j} = t_{i,j}$ and $t_{i,j+n} = t_{i,j}$ for every $i, j \geq 1$.

Let $\mathcal{D}$ be the collection of all domino systems, $\mathcal{D}_{\int}$ be the subset of $\mathcal{D}$ that admit a solution and $\mathcal{D}_{\surd\int}$ be the subset of $\mathcal{D}$ that admit a periodic solution. Note that $\mathcal{D}_{\surd\int} \subseteq \mathcal{D}_{\int}$. It is well-known [3, Theorem 3.1.7] that the sets $\mathcal{D} \setminus \mathcal{D}_{\int}$ and $\mathcal{D}_{\surd\int}$ are *recursively inseparable*, that is, there is no recursive (i.e. decidable) subset $\mathcal{D}' \subseteq \mathcal{D}$ of domino systems such that $\mathcal{D}_{\surd\int} \subseteq \mathcal{D}' \subseteq \mathcal{D}_{\int}$.

We use this property in our reduction. For every domino system D, we construct a signature $\mathbf{S} = \mathbf{S}(D)$, an ontology $\mathcal{Q} = \mathcal{Q}(D)$ which is an $\mathcal{ALC}$-TBox, and an axiom $\alpha \in \mathcal{Q}$ such that:

(a) if D does not have a solution then α is not $\mathbf{S}$-essential in $\mathcal{Q}$ w.r.t. L, and
(b) if D has a periodic solution then α is $\mathbf{S}$-essential in $\mathcal{Q}$.

In other words, for the set $\mathcal{D}'$ of domino systems D such that α is $\mathbf{S}$-essential in $\mathcal{Q} = \mathcal{Q}(D)$ w.r.t. L, we have $\mathcal{D}_{\surd\int} \subseteq \mathcal{D}' \subseteq \mathcal{D}_{\int}$. Since $\mathcal{D} \setminus \mathcal{D}_{\int}$ and $\mathcal{D}_{\surd\int}$ are recursively inseparable, this implies undecidability for $\mathcal{D}'$ and hence for the problem of checking $\mathbf{S}$-essential axioms, because otherwise one can use this problem for deciding membership in $\mathcal{D}'$.

The signature $\mathbf{S}$, ontology $\mathcal{Q}$ and axiom $\alpha \in \mathcal{Q}$ are constructed as follows. Given a domino system $D = (T, H, V)$, let $\mathbf{S}$ consist of fresh atomic concepts A_t for every $t \in T$ and two atomic roles r_H and r_V. We define $\mathcal{Q}$ to consists of axioms (q_1)–(q_5) from Figure 6.2 and set α to be the axiom (q_5).

Axioms of form (q_1)–(q_4) express that every domain element in a model for $\mathcal{Q}$ is assigned with a unique tile $t \in T$ and has horizontal and vertical matching successors. Axiom (q_5) plays a special role in our reduction for excluding those models

$$\begin{array}{lr}
(q_1)\ \top \sqsubseteq A_{t_1} \sqcup \cdots \sqcup A_{t_n} & \text{if } T = \{t_1, \ldots, t_n\} \\
(q_2)\ A_{t_i} \sqcap A_{t_j} \sqsubseteq \bot & \text{whenever } t_i \neq t_j, \\
(q_3)\ A_{t_i} \sqsubseteq \exists r_H.(\bigsqcup_{(t_i,t_j)\in H} A_{t_j}\,) & t_i, t_j \in T \\
(q_4)\ A_{t_i} \sqsubseteq \exists r_V.(\bigsqcup_{(t_i,t_j)\in V} A_{t_j}\,) & \\
\hline
(q_5)\ \top \sqsubseteq \exists \boldsymbol{s}.[\exists r_H.\exists r_V.\boldsymbol{B} \sqcap \exists r_V.\exists r_H.\neg\boldsymbol{B}] & =: \alpha
\end{array}$$

Fig. 6.2. An ontology $\mathcal{Q}$ for a domino system D

of $\mathcal{Q}$ for which the horizontal and vertical matching relations do not commute. We can show that all axioms from $\mathcal{Q}$ are *independent*, i.e. none of the axioms is a logical consequence of the remaining axioms. In the remainder, we prove properties (a) and (b) formulated above.

In order to prove property (a), assume that α is $\mathbf{S}$-essential in $\mathcal{Q}$ w.r.t. L. We demonstrate that D has a solution in this case.

Let $\mathcal{Q}^\alpha$ be a minimal $\mathbf{S}$-module in $\mathcal{Q}$ containing α. Note that $\mathcal{Q}^\alpha$ implies all axioms of form (q_1)–(q_4) in $\mathcal{Q}$, since the signature of these axioms is a subset of $\mathbf{S}$. Since $\mathcal{Q}^\alpha$ contains α and all axioms of $\mathcal{Q}$ are independent, this is only possible when $\mathcal{Q}^\alpha = \mathcal{Q}$.

Since $\mathcal{Q}^\alpha = \mathcal{Q}$ is a minimal $\mathbf{S}$-module in $\mathcal{Q}$, the set $\mathcal{Q}_1 := \mathcal{Q} \setminus \{\alpha\}$ is not an $\mathbf{S}$-module in $\mathcal{Q}$, and so, by the part 2 of Proposition 4, $\mathcal{Q}$ is not a model $\mathbf{S}$-conservative extension of $\mathcal{Q}_1$. This means that there is an $\mathbf{S}$-interpretation $\mathcal{I}_1 = (\Delta, \cdot^{\mathcal{I}_1})$ that is a model of the axioms of form (q_1)–(q_4), but which cannot be expanded to a model of α by interpreting atomic role s and atomic concept B. We claim that this is possible only if relations r_H and r_V commute in $\mathcal{I}_1$, that is, whenever $r_H(a,b)$, $r_V(b,c_1)$, $r_V(a,d)$ and $r_H(d,c_2)$ hold in $\mathcal{I}_1$, then it must be the case that $c_1 = c_2$. Indeed, otherwise one can expand $\mathcal{I}_1$ to a model $\mathcal{I}$ of α by setting $s^{\mathcal{I}} = \{(x,a) \mid x \in \Delta\}$ and $B^{\mathcal{I}} = \{c_1\}$. Since $\mathcal{I}$ satisfies all formulas of forms (q_1)–(q_4) and admits commutativity property for relations r_H and r_V, we can see that D has a solution.

In order to prove property (b), assume that D has a periodic solution $t_{i,j}$ with the periods $m, n \geq 1$. We demonstrate that α is $\mathbf{S}$-essential in $\mathcal{Q}$ by showing that $\mathcal{Q}_1 := \mathcal{Q} \setminus \{\alpha\}$ is not an $\mathbf{S}$-module in $\mathcal{Q}$. For this purpose we construct an $\mathcal{ALCO}$-ontology $\mathcal{P}$ such that $\mathcal{P} \cup \mathcal{Q} \models \bot$, but $\mathcal{P} \cup \mathcal{Q}_1 \not\models \bot$. We define $\mathcal{P}$ such that every model of $\mathcal{P}$ is a finite encoding of the periodic solution $t_{i,j}$. For every pair (i,j) with $1 \leq i \leq m$ and $1 \leq j \leq n$ we introduce a fresh individual $a_{i,j}$ and add the following axioms to $\mathcal{P}$

$$\begin{array}{ll}
(p_1)\ a_{i,j} : A_{t_{i,j}}, & (p_4)\ \top \sqsubseteq \bigsqcup_{1\leq i\leq m,\, 1\leq j\leq n} \{a_{i,j}\}, \\
(p_2)\ r_V(a_{i_1,j}, a_{i_2,j}), & (p_5)\ \{a_{i_1,j}\} \sqsubseteq \forall r_V.\{a_{i_2,j}\},\ i_2 = i_1 + 1 \mod m \\
(p_3)\ r_H(a_{i,j_1}, a_{i,j_2}), & (p_6)\ \{a_{i,j_1}\} \sqsubseteq \forall r_H.\{a_{i,j_2}\},\ j_2 = j_1 + 1 \mod n
\end{array}$$

The axioms (p_1)–(p_4) encode the solution $t_{i,j}$ for $\mathcal{D}$, and so, ensure that axioms (q_1)–(q_4) are satisfied. The axioms (p_5) and (p_6) ensure that the relations r_V and r_H are defined completely, i.e. no other relations except for those specified in the first column hold in models of $\mathcal{P}$. In particular, in every model of $\mathcal{P}$, relations r_H and

r_V commute, and so, axiom α is not satisfied. Consequently, $\mathcal{P} \cup \mathcal{Q}$ is unsatisfiable, whereas $\mathcal{P} \cup \mathcal{Q}_1$ is satisfiable, and so, $\mathcal{Q}_1$ is not an $\mathbf{S}$-module in $\mathcal{Q}$. $\diamond$

Corollary 1. *There exists no algorithm for performing any of the tasks T1-T3 from* (6.3)*, and* (6.4) *for $\mathcal{ALCO}$-ontologies.*

Proof. Theorem 1 implies directly that there is no algorithm for task T3 from (6.4), because otherwise, one can check if an axiom α is $\mathbf{S}$-essential in $\mathcal{Q}$ by simply computing the set of all essential axioms by this algorithm for T3 and then checking if α is contained in this set. The remaining tasks from (6.3) are unsolvable since they are reducible to T3 by Proposition 3. $\diamond$

Corollary 2. *Given a signature* $\mathbf{S}$*, an $\mathcal{ALC}$-ontology $\mathcal{Q}$ and an ontology $\mathcal{Q}_1 \subseteq \mathcal{Q}$, it is undecidable whether $\mathcal{Q}_1$ is an* $\mathbf{S}$*-module in $\mathcal{Q}$ w.r.t.* $\mathrm{L} = \mathcal{ALCO}$.

Proof. The procedure for deciding if $\mathcal{Q}_1$ is an $\mathbf{S}$-module in $\mathcal{Q}$ can be used for solving task T1, which is not possible by Corollary 1. Indeed, by enumerating the subsets of $\mathcal{Q}$ and checking if they are modules, one can compute all subsets $\mathcal{M}$ of $\mathcal{Q}$ that are $\mathbf{S}$-modules in $\mathcal{Q}$. The set of all minimal modules in $\mathcal{Q}$ can be then computed from $\mathcal{M}$ by filtering out those sets in $\mathcal{M}$ that are proper subsets of some other sets in $\mathcal{M}$. $\diamond$

Corollary 2 has a strong impact on the problem of knowledge reuse and forces us to revisit the original problem we aim at solving. As the problem of extracting minimal modules cannot be computationally solved for OWL DL in none of the forms T1-T3, T1s or T2s, we propose to relax some of the requirements in these tasks. We cannot drop the requirements that extracted fragments should be modules since, in this case, we have no guarantee for the correctness of the result. We can sacrifice, however, the minimality requirements for the computed modules and consider the following weakened version of the task T2:

T2w. compute *some* small enough $\mathbf{S}$-module in $\mathcal{Q}$ (6.6)

Although it is always possible to extract an $\mathbf{S}$-module in $\mathcal{Q}$ (one can simply return $\mathcal{Q}$ which is always an $\mathbf{S}$-module in $\mathcal{Q}$), it still makes sense to develop, compare, and practically apply procedures that compute reasonably small modules. In what follows, we describe two procedures of this form, based on the notions of locality, which we first introduced in [4]. The modules we obtain might be larger than the minimal modules and therefore we need to show that, in practice, they are still reasonably small.

6.5 Modules Based on Locality

In this section, we formulate the notion of locality, first introduced in [4] which will constitute the basis of our algorithm for extracting modules. In this section, we restrict ourselves to $\mathcal{SHIQ}$, although the results provided here could we extended to $\mathcal{SHOIQ}$ using the same treatment for nominals as in [4].

6.5.1 Locality

As a consequence of Case 2 in Proposition 4, model conservative extensions can be used as a sufficient condition for the notion of module. It is not possible, however, to design a procedure that extracts modules based on this condition since the problem of deciding model conservative extensions is highly undecidable [12]. The idea underlying this notion, however, can be used to establish sufficient conditions for the notion of module which are decidable and can be used in practice.

Consider the first part of Example 1, where we show that the set $\mathcal{Q}$ of axioms M1-M5 in Figure 6.1 is a model $\mathbf{S}$-conservative extension of $\mathcal{Q}_1 = \{\mathcal{M}1, \ldots, \mathcal{M}4\}$, for $\mathbf{S} = \{\mathsf{Cystic\_Fibrosis},\ \mathsf{Genetic\_Disorder}\}$. In this example, the model conservative extension was shown by finding expansions of $\mathsf{Sig}(\mathcal{Q}_1)$-interpretations to models of $\mathcal{Q}$ in which all concept and atomic roles not in $\mathsf{Sig}(\mathcal{Q}_1)$ were interpreted as the empty set. One could consider the cases where conservative extensions (and hence modules) can be determined in this manner. This idea can be formalized using the notion of locality:

Definition 4 (Locality [4]). *Let* $\mathbf{S}$ *be a signature. We say that an* axiom α is local w.r.t. $\mathbf{S}$ *if every trivial expansion of any* $\mathbf{S}$*-interpretation to* $\mathbf{S} \cup \mathsf{Sig}(\alpha)$ *is a model of* α*. We denote by* $\mathsf{local}(\mathbf{S})$ *the collection of all axioms that are local w.r.t.* $\mathbf{S}$*. An ontology* $\mathcal{O}$ *is local w.r.t.* $\mathbf{S}$ *if* $\mathcal{O} \subseteq \mathsf{local}(\mathbf{S})$. $\diamond$

Intuitively, an ontology $\mathcal{O}$ is *local* w.r.t. a signature $\mathbf{S}$ if we can take *any* interpretation for the symbols in $\mathbf{S}$ and extend it to a *model* of $\mathcal{O}$ that interprets the additional symbols as the empty set.

Example 3. Axiom M5 is local for $\mathbf{S} = \{\mathsf{Cystic\_Fibrosis},\ \mathsf{Genetic\_Disorder}\}$. Indeed, as shown in Example 1, for every trivial expansion $\mathcal{I}$ of an $\mathbf{S}$-interpretation to $\mathbf{S} \cup \mathsf{Sig}(\alpha)$, the concept $\mathsf{DEFBI\_Gene}$ is interpreted as the empty set, and so, $\mathcal{I}$ satisfies M5.

On the other hand, M5 is not local w.r.t. $\mathbf{S} = \{\mathsf{DEFBI\_Gene}\}$. Indeed, take any $\mathbf{S}$-interpretation $\mathcal{I}_1$ in which $\mathsf{DEFBI\_Gene}$ is interpreted as a non-empty set. Then, for every trivial expansion $\mathcal{I}$ of $\mathcal{I}_1$, the concept on the left-hand-side of M5 is always interpreted as a non-empty set, whereas the concept on the right-hand-side is always interpreted as the empty set. So $\mathcal{I}$ does not satisfy α.

In fact, this shows that axiom M5 is local w.r.t. $\mathbf{S}$ if and only if $\mathbf{S}$ does not contain $\mathsf{DEFBI\_Gene}$. $\diamond$

The following is a simple but useful property of locality shows that the set of local axioms can only become smaller if the signature expands:

Lemma 2 (Anti-Monotonicity of Locality). *Let* $\mathbf{S}_1$ *and* $\mathbf{S}_2$ *be signature sets. Then* $\mathbf{S}_1 \subseteq \mathbf{S}_2$ *implies* $\mathsf{local}(\mathbf{S}_2) \subseteq \mathsf{local}(\mathbf{S}_1)$.

Proof. Let $\alpha \in \mathsf{local}(\mathbf{S}_2)$. We demonstrate that $\alpha \in \mathsf{local}(\mathbf{S}_1)$. For this purpose, let $\mathcal{I}_1$ be an arbitrary $\mathbf{S}_1$-interpretation. We need to show that every trivial expansion $\mathcal{I}_1'$ of $\mathcal{I}_1$ to $\mathbf{S}_1 \cup \mathsf{Sig}(\alpha)$ is a model of α.

Let $\mathcal{I}_2$ be a trivial expansion of $\mathcal{I}_1$ to $\mathbf{S}_2$ (note that $\mathbf{S}_1 \subseteq \mathbf{S}_2$). Since $\alpha \in \mathsf{local}(\mathbf{S}_2)$, every trivial expansion $\mathcal{I}_2'$ of $\mathcal{I}_2$ to $\mathbf{S}_2 \cup \mathsf{Sig}(\alpha)$ is a model of α. Note that $\mathcal{I}_2'$ is a trivial expansion of $\mathcal{I}_1$ to $\mathbf{S}_2 \cup \mathsf{Sig}(\alpha)$, hence $\mathcal{I}_1' = \mathcal{I}_2'|_{\mathbf{S}_1 \cup \mathsf{Sig}(\alpha)} \models \alpha$. $\diamond$

Locality can be used to formulate a sufficient condition for an ontology to be a model conservative extension of another ontology:

Proposition 5 (Locality $\Rightarrow$ Model Conservativity). *Let $\mathcal{O}_1$, $\mathcal{O}_2$ be two ontologies and $\mathbf{S}$ a signature such that $\mathcal{O}_2$ is local w.r.t.* $\mathbf{S} \cup \mathsf{Sig}(\mathcal{O}_1)$. *Then $\mathcal{O}_1 \cup \mathcal{O}_2$ is an $\mathbf{S}$-model conservative extension of $\mathcal{O}_1$.*

Proof. Let $\mathcal{I}_1$ be a model of $\mathcal{O}_1$. We show that there exists a model $\mathcal{I}$ of $\mathcal{O}_1 \cup \mathcal{O}_2$ such that $\mathcal{I}|_{\mathbf{S}} = \mathcal{I}_1|_{\mathbf{S}}$.

Let $\mathcal{I}$ be a trivial expansion of $\mathcal{I}_1|_{\mathbf{S} \cup \mathsf{Sig}(\mathcal{O}_1)}$ to $\mathbf{S} \cup \mathsf{Sig}(\mathcal{O}_1) \cup \mathsf{Sig}(\mathcal{O}_2)$, thus, in particular, $\mathcal{I}|_{\mathbf{S} \cup \mathsf{Sig}(\mathcal{O}_1)} = \mathcal{I}_1|_{\mathbf{S} \cup \mathsf{Sig}(\mathcal{O}_1)}$. We need to show that $\mathcal{I}$ is a model of $\mathcal{O}_1 \cup \mathcal{O}_2$. Since $\mathcal{O}_2$ is local w.r.t. $\mathbf{S} \cup \mathsf{Sig}(\mathcal{O}_1)$, by Definition 4, $\mathcal{I}$ is a model of $\mathcal{O}_2$. Moreover, since $\mathcal{I}|_{\mathsf{Sig}(\mathcal{O}_1)} = \mathcal{I}_1|_{\mathsf{Sig}(\mathcal{O}_1)}$ and $\mathcal{I}_1 \models \mathcal{O}_1$, we have $\mathcal{I} \models \mathcal{O}_1$. Hence, $\mathcal{I} \models \mathcal{O}_1 \cup \mathcal{O}_2$ what was required to show. $\diamond$

Using Proposition 5 and Property 2 of Proposition 4 we obtain:

Corollary 3. *Let $\mathcal{O}_1$, $\mathcal{O}_2$ and* $\mathbf{S}$ *be as given in Proposition 5. Then $\mathcal{O}_1$ is an* $\mathbf{S}$*-module in $\mathcal{O}_1 \cup \mathcal{O}_2$.*

Corollary 3 suggests how one can use locality for extracting modules. Given an ontology $\mathcal{Q}$ and a signature $\mathbf{S}$, it is sufficient to partition $\mathcal{Q}$ into $\mathcal{Q}_1 \cup \mathcal{Q}_2$ such that $\mathcal{Q}_2$ is local w.r.t. $\mathbf{S} \cup \mathsf{Sig}(\mathcal{Q}_1)$. In this case, $\mathcal{Q}_1$ is an $\mathbf{S}$-module in $\mathcal{Q}$.

Definition 5 (Modules based on Locality Condition)
Given an ontology $\mathcal{Q}$ and a signature $\mathbf{S}$, *we say that $\mathcal{Q}_1 \subseteq \mathcal{Q}$* is a locality-based $\mathbf{S}$-module in $\mathcal{Q}$ *if $\mathcal{Q} \setminus \mathcal{Q}_1$ is local w.r.t* $\mathbf{S} \cup \mathsf{Sig}(\mathcal{Q}_1)$. $\diamond$

Remark 1. Note from Definition 5 that every locality-based $\mathbf{S}$-module $\mathcal{Q}_1$ in $\mathcal{Q}$, is also a locality-based $\mathbf{S} \cup \mathsf{Sig}(\mathcal{Q}_1)$-module in $\mathcal{Q}$. $\diamond$

Remark 2. Note that $\mathcal{Q}_1$ is a locality-based $\mathbf{S}$-module in $\mathcal{Q}$ if every trivial expansion of every model of $\mathcal{Q}_1$ based on $\mathbf{S} \cup \mathsf{Sig}(\mathcal{Q}_1)$ to $\mathbf{S} \cup \mathsf{Sig}(\mathcal{Q})$, is a model for $\mathcal{Q}$. $\diamond$

Example 4 (Example 3, continued). We have seen in Example 3 that axiom M5 is local w.r.t. every $\mathbf{S}$ that does not contain the atomic concept $\mathsf{DEFBI\_Gene}$. In particular, for $\mathcal{Q}_1$ consisting of axioms M1-M4 from Figure 6.1, M5 is local w.r.t. $\mathsf{Sig}(\mathcal{Q}_1)$. Hence, according to Definition 5, $\mathcal{Q}_1$ is a locality-based $\mathbf{S}$-module in $\mathcal{Q} = \{\mathcal{M}1, \ldots, \mathcal{M}5\}$ for every $\mathbf{S} \subseteq \mathsf{Sig}(\mathcal{Q}_1)$. $\diamond$

Remark 3. Note that the analog of the Part 1 in Proposition 1 does not hold for locality-based modules since locality-based modules are not necessarily upward-closed. For example, consider the following ontology and a signature:

$$\mathcal{Q} = \{(1)\ A_1 \sqsubseteq A_2;\ (2)\ B \sqsubseteq A_1;\ (3)\ B \sqsubseteq A_2\} \quad \mathbf{S} = \{A_1, A_2\}$$

It is easy to see that the set $\mathcal{Q}_1 = \{A_1 \sqsubseteq A_2\}$ consisting of the first axiom from $\mathcal{Q}$ is a locality-based $\mathbf{S}$-module in $\mathcal{Q}$, since both axioms (2) and (3) are local w.r.t. $\mathbf{S} \cup \mathsf{Sig}(\mathcal{Q}_1) = \{A_1, A_2\}$. However, its superset $\mathcal{Q}'_1 = \{A_1 \sqsubseteq A_2;\ B \sqsubseteq A_1\}$ is not a locality-based module w.r.t. $\mathbf{S}$, since the axiom $B \sqsubseteq A_2$ in $\mathcal{Q} \setminus \mathcal{Q}'_1$ is not local w.r.t. $\mathbf{S} \cup \mathsf{Sig}(\mathcal{Q}'_1) = \{A_1, A_2, B\}$. Note that $\mathcal{Q}'_1$ is an $\mathbf{S}$-module in $\mathcal{Q}$, since it is a superset of an $\mathbf{S}$-module $\mathcal{Q}_1$. $\diamondsuit$

6.5.2 Testing Locality

As demonstrated in Example 3, for testing locality of an axiom α w.r.t. $\mathbf{S}$, it is sufficient to interpret every atomic concept and atomic role not in $\mathbf{S}$ with the empty set and then check if α is satisfied for all interpretations of the remaining symbols. This observation suggests that locality can be tested by first simplifying the ontology by eliminating atomic roles and concepts that are not in $\mathbf{S}$, and then checking if the resulting axioms are satisfied in every interpretation for the remaining symbols. This idea is formalized as follows:

Proposition 6 (Testing Locality). *Let $\mathcal{O}$ be a $\mathcal{SHIQ}$ ontology and* $\mathbf{S}$ *a signature. Let $\mathcal{O}_\mathbf{S}$ be obtained from $\mathcal{O}$ by applying the transformations below, where every A is an atomic concept, every r is an atomic role with $A, r \notin \mathbf{S}$, and every R is a role r or r^- with $r \notin \mathbf{S}$: (1) replace all concepts of form A, $\exists R.C$ or $(\geqslant n\, R.C)$ with $\bot$; (2) remove every transitivity axiom* $\mathsf{Trans}(r)$*; (3) replace every assertion $a : A$ and $r(a, b)$ with the contradiction axiom $\top \sqsubseteq \bot$.*

Then $\mathcal{O}$ is local w.r.t. $\mathbf{S}$ *iff every axiom in $\mathcal{O}_\mathbf{S}$ is a tautology.*

Proof. It is easy to check that the transformation above preserves the satisfaction of axioms under every trivial expansion $\mathcal{I}$ of every $\mathbf{S}$-interpretation to $\mathbf{S} \cup \mathsf{Sig}(\mathcal{O})$. Hence, the resulting ontology $\mathcal{O}_\mathbf{S}$ is local w.r.t. $\mathbf{S}$ iff the original ontology $\mathcal{O}$ was local w.r.t. $\mathbf{S}$. Moreover, it is easy to see that there are no atomic concepts and atomic roles outside $\mathbf{S}$ left in $\mathcal{O}_\mathbf{S}$ after the transformation. Hence, every axiom α from $\mathcal{O}_\mathbf{S}$ is a tautology iff $\mathcal{Q}$ is local w.r.t. $\mathbf{S}$. $\diamondsuit$

Note that according to Definition 4, assertions $a : A$ and $r(a, b)$ can never be local since they can only be satisfied by interpretations that interpret A and r as non-empty sets. Hence, assertions must be included in every locality-based module, which is reflected in the step (3) of the transformation in Proposition 6.

Example 5. Recall that in Example 3 we have demonstrated that axiom M5 from Figure 6.1 is local w.r.t. $\mathbf{S} = \{\mathsf{Cystic\_Fibrosis},\ \mathsf{Genetic\_Disorder}\}$. Now we demonstrate this using Proposition 6. Indeed, according to this proposition we need to perform the following replacements:

$$\begin{aligned}
\mathsf{DEFBI\_Gene} &\Rightarrow \bot && \textit{(by (1) since } \mathsf{DEFBI\_Gene} \notin \mathbf{S}\textit{)}\\
\mathsf{Immuno\_Protein\_Gene} &\Rightarrow \bot && \textit{(by (1) since } \mathsf{Immuno\_Protein\_Gene} \notin \mathbf{S}\textit{)}\\
\exists \mathsf{associated\_With}.\mathsf{Cystic\_Fibrosis} &\Rightarrow \bot && \textit{(by (1) since } \mathsf{associated\_With} \notin \mathbf{S}\textit{)}
\end{aligned}$$

Hence, axiom M5 will be translated to axiom $\bot \sqsubseteq \bot \sqcap \bot$ which is a tautology. $\diamondsuit$

An important conclusion of Proposition 6 is that one can use the standard capabilities of available DL-reasoners[2] such as FaCT++ [19], RACER [13], Pellet [18] or KAON2 [14] for testing locality since these reasoners can test for DL-tautologies. Checking for tautologies in description logics is, theoretically, a difficult problem (e.g. for DL $\mathcal{SHIQ}$ is NEXPTIME-complete). There are, however, several reasons to believe that the locality test would perform well in practice. First, and most importantly, the size of the axioms in an ontology is usually small compared to the size of the ontology. Second, DL reasoners are highly optimized for standard reasoning tasks and behave well for most realistic ontologies.

In case this is too costly, it is possible to formulate a tractable approximation to the locality conditions for $\mathcal{SHIQ}$:

Definition 6 (Syntactic Locality for $\mathcal{SHIQ}$). *Let* $\mathbf{S}$ *be a signature. The following grammar recursively defines two sets of concepts* $\mathcal{C}^{\perp}_{\mathbf{S}}$ *and* $\mathcal{C}^{\top}_{\mathbf{S}}$ *for a signature* $\mathbf{S}$*:*

$$\begin{aligned}\mathcal{C}^{\perp}_{\mathbf{S}} &::= A^{\perp} \mid (\neg C^{\top}) \mid (C \sqcap C^{\perp}) \mid (\exists R^{\perp}.C) \\ &\quad\mid (\exists R.C^{\perp}) \mid ({\geqslant}\, n\, R^{\perp}.C) \mid ({\geqslant}\, n\, R.C^{\perp})\,. \\ \mathcal{C}^{\top}_{\mathbf{S}} &::= (\neg C^{\perp}) \mid (C^{\top}_1 \sqcap C^{\top}_2)\,.\end{aligned}$$

where $A^{\perp} \notin \mathbf{S}$ *is a atomic concept,* R *is a role, and* C *is a concept,* $C^{\perp} \in \mathcal{C}^{\perp}_{\mathbf{S}}$*,* $C^{\top}_{(i)} \in \mathcal{C}^{\top}_{\mathbf{S}}$*,* $i = 1, 2$*, and* $R^{\perp} \notin \mathsf{Rol}(\mathbf{S})$ *is a role.*

An axiom α *is* syntactically local w.r.t. $\mathbf{S}$ *if it is of one of the following forms:* (1) $R^{\perp} \sqsubseteq R$*, or* (2) $\mathsf{Trans}(R^{\perp})$*, or* (3) $C^{\perp} \sqsubseteq C$ *or* (4) $C \sqsubseteq C^{\top}$*. We denote by* $\mathsf{s\_local}(\mathbf{S})$ *the set of all* $\mathcal{SHIQ}$*-axioms that are syntactically local w.r.t.* $\mathbf{S}$*. A* $\mathcal{SHIQ}$*-ontology* $\mathcal{O}$ *is* syntactically local w.r.t. $\mathbf{S}$ *if* $\mathcal{O} \subseteq \mathsf{s\_local}(\mathbf{S})$. $\diamond$

Intuitively, every concept in $\mathcal{C}^{\perp}_{\mathbf{S}}$ becomes equivalent to $\perp$ if we replace every symbol $A^{\perp}$ or $R^{\perp}$ not in $\mathbf{S}$ with the bottom concept $\perp$ and the empty role respectively, which are both interpreted as the empty set under every interpretation. Similarly, the concepts from $\mathcal{C}^{\top}_{\mathbf{S}}$ are equivalent to $\top$ under this replacement. Syntactically local axioms become tautologies after these replacements.

For example, it is easy to show that the axiom M2 from Figure 6.1 is local w.r.t. $\mathbf{S} = \{\mathsf{Fibrosis}, \mathsf{has\_Origin}\}$: if we replace the remaining symbols in this axiom with $\perp$, we obtain a tautology $\perp \equiv \perp$:

$$\overbrace{\mathsf{Genetic\_Fibrosis}}^{\perp} \equiv \underline{\mathsf{Fibrosis}} \sqcap \underbrace{\exists \underline{\mathsf{has\_Origin}}.\overbrace{\mathsf{Genetic\_Origin}}^{\perp}}_{\perp}$$

To distinguish the original notion of locality from its syntactic approximation, we sometimes call the first as *semantic locality*, as it is defined in terms of the interpretations.

[2] See `http://www.cs.man.ac.uk/~sattler/reasoners.html` for a list of currently available reasoners.

It is easy to show that the analog of Lemma 2 also holds for syntactic locality:

Lemma 3 (Anti-Monotonicity of Syntactic Locality)
Let $\mathbf{S}_1$ *and* $\mathbf{S}_2$ *be signature sets. Then* $\mathbf{S}_1 \subseteq \mathbf{S}_2$ *implies* $\mathsf{s\_local}(\mathbf{S}_2) \subseteq \mathsf{s\_local}(\mathbf{S}_1)$.

Proof. It is easy to see from Definition 6 that $\mathcal{C}^{\perp}_{\mathbf{S}_2} \sqsubseteq \mathcal{C}^{\perp}_{\mathbf{S}_1}$, $\mathcal{C}^{\top}_{\mathbf{S}_2} \sqsubseteq \mathcal{C}^{\top}_{\mathbf{S}_1}$, and hence, $\mathsf{s\_local}(\mathbf{S}_2) \sqsubseteq \mathsf{s\_local}(\mathbf{S}_1)$. ◇

As expected, syntactic locality is an approximation for semantic locality:

Proposition 7. *Let* $\mathbf{S}$ *be a signature. Then* $\mathsf{s\_local}(\mathbf{S}) \subseteq \mathsf{local}(\mathbf{S})$.

Proof. Let α be an axiom that is syntactically local w.r.t. $\mathbf{S}$ and let $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$ be a trivial expansion of some $\mathbf{S}$-interpretation to $\mathbf{S} \cup \mathsf{Sig}(\alpha)$. We have to demonstrate that $\mathcal{I}$ is a model of α. By induction over the definitions of $\mathcal{C}^{\perp}_{\mathbf{S}}$ and $\mathcal{C}^{\top}_{\mathbf{S}}$ from Definition 6, it is easy to show that: (i) every role $R \notin \mathsf{Rol}(\mathbf{S})$ and every every concept from $\mathcal{C}^{\perp}_{\mathbf{S}}$ is interpreted in $\mathcal{I}$ with the empty set, and (ii) every concept from $\mathcal{C}^{\top}_{\mathbf{S}}$ is interpreted in $\mathcal{I}$ with Δ. By checking all the possible cases for a syntactically local axiom α in Definition 5, it is easy to see that in every of these cases $\mathcal{I}$ is a model of α. ◇

The converse of Proposition 7 does not hold in general since there are semantically local axioms that are not syntactically local. For example, the axiom $\alpha = (A \sqsubseteq A \sqcup B)$ is a tautology and thus is local w.r.t. every $\mathbf{S}$. This axiom, however, is not syntactically local w.r.t. $\mathbf{S} = \{A, B\}$ since it involves symbols in $\mathbf{S}$ only. Another example, which is not a tautology, is the GCI $\alpha = (\exists R.\neg A \sqsubseteq \exists R.\neg B)$, which is semantically local w.r.t. $\mathbf{S} = \{R\}$ ($\exists R.\top \sqsubseteq \exists R.\top$ is a tautology), but not syntactically

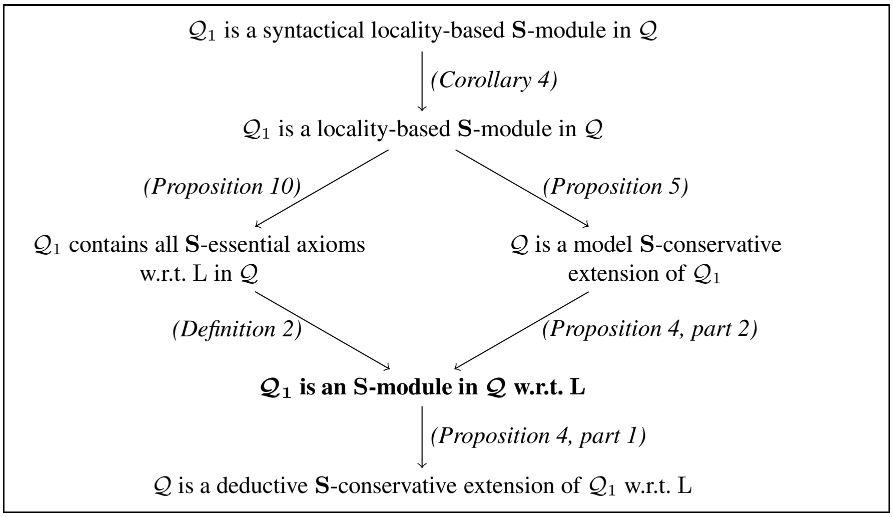

$\mathcal{Q}_1$ is a syntactical locality-based $\mathbf{S}$-module in $\mathcal{Q}$
(Corollary 4)
$\mathcal{Q}_1$ is a locality-based $\mathbf{S}$-module in $\mathcal{Q}$
(Proposition 10)
(Proposition 5)
$\mathcal{Q}_1$ contains all $\mathbf{S}$-essential axioms w.r.t. L in $\mathcal{Q}$
$\mathcal{Q}$ is a model $\mathbf{S}$-conservative extension of $\mathcal{Q}_1$
(Definition 2)
(Proposition 4, part 2)
$\mathcal{Q}_1$ is an $\mathbf{S}$-module in $\mathcal{Q}$ w.r.t. L
(Proposition 4, part 1)
$\mathcal{Q}$ is a deductive $\mathbf{S}$-conservative extension of $\mathcal{Q}_1$ w.r.t. L

Fig. 6.3. Summary for the main theoretical results of the chapter

local. Thus, the limitation of syntactic locality is its inability to perform reasoning on elements from $\mathbf{S}$.

We distinguish the notion of modules based on these two locality conditions as *semantic locality-based modules* and *syntactic locality-based modules*.

Corollary 4. *If $\mathcal{Q}_1$ is a syntactic locality-based $\mathbf{S}$-module in $\mathcal{Q}$, then $\mathcal{Q}_1$ is a semantic locality-based $\mathbf{S}$-module in $\mathcal{Q}$.*

For the reference and for the convenience of the reader, we illustrate in Figure 6.3 the relationships between the key theoretical results in this chapter.

6.5.3 Computing Locality-Based Modules

Recall that, according to Definition 5, in order to construct a locality-based $\mathbf{S}$-module in an ontology $\mathcal{Q}$, it suffices to partition the ontology $\mathcal{Q}$ as $\mathcal{Q} = \mathcal{Q}_1 \cup \mathcal{Q}_2$ such that $\mathcal{Q}_2$ is local w.r.t. $\mathbf{S} \cup \mathsf{Sig}(\mathcal{Q}_1)$. Algorithm 1 outlines a simple procedure which performs this task. Given an effective locality test locality_test($\alpha, \mathbf{S}$) (which uses either a reasoner or the syntactical approximation) which returns true only if the axiom α is local w.r.t. $\mathbf{S}$, the algorithm first initializes the partition to the trivial one: $\mathcal{Q}_1 = \emptyset$ and $\mathcal{Q}_2 = \mathcal{Q}$, and then repeatedly moves to $\mathcal{Q}_1$ those axioms from $\mathcal{Q}_2$ that are not local w.r.t. $\mathbf{S} \cup \mathsf{Sig}(\mathcal{Q}_1)$ until no such axioms are left in $\mathcal{Q}_2$.

In Table 6.2 we provide a trace of Algorithm 1 for the input $(\mathcal{Q}, \mathbf{S})$, where $\mathcal{Q}$ is an ontology consisting of the axioms M1-M5 from Figure 6.1 and $\mathbf{S}$ is a signature $\mathbf{S} = \{\mathsf{Cystic\_Fibrosis}, \mathsf{Genetic\_Disorder}\}$. Each row in the table corresponds to an iteration of the while loop in Algorithm 1. The last column of the table provides the results of the locality test in line 4. Note that the syntactic locality condition was sufficient in all tests: all axioms that were semantically non-local were also syntactically non-local.

Algorithm 1. extract_module($\mathcal{Q}, \mathbf{S}$)

Input:
$\mathcal{Q}$: ontology
$\mathbf{S}$: signature
Output:
$\mathcal{Q}_1$: a locality-based $\mathbf{S}$-module in $\mathcal{Q}$

```
 1: $\mathcal{Q}_1 \leftarrow \emptyset \quad \mathcal{Q}_2 \leftarrow \mathcal{Q}$
 2: while not empty($\mathcal{Q}_2$) do
 3:     $\alpha \leftarrow$ select_axiom($\mathcal{Q}_2$)
 4:     if locality_test( $\alpha$, $\mathbf{S} \cup \mathsf{Sig}(\mathcal{Q}_1)$ ) then
 5:         $\mathcal{Q}_2 \leftarrow \mathcal{Q}_2 \setminus \{\alpha\}$ {$\alpha$ is processed}
 6:     else
 7:         $\mathcal{Q}_1 \leftarrow \mathcal{Q}_1 \cup \{\alpha\}$ {move $\alpha$ into $\mathcal{Q}_1$}
 8:         $\mathcal{Q}_2 \leftarrow \mathcal{Q} \setminus \mathcal{Q}_1$ {reset $\mathcal{Q}_2$ to the complement of $\mathcal{Q}_1$}
 9:     end if
10: end while
11: return $\mathcal{Q}_1$
```

Table 6.2. A trace of Algorithm 1 for the input $\mathcal{Q} = \{\mathcal{M}1, \ldots, \mathcal{M}5\}$ and $\mathbf{S} = \{\mathsf{Cystic\_Fibrosis},\ \mathsf{Genetic\_Disorder}\}$

	$\mathcal{Q}_1$	$\mathcal{Q}_2$	New elements in $\mathbf{S} \cup \mathsf{Sig}(\mathcal{Q}_1)$	α	local?
1	–	$\mathcal{M}1 - \mathcal{M}5$	Cystic_Fibrosis, Genetic_Disorder	$\mathcal{M}1$	No
2	$\mathcal{M}1$	$\mathcal{M}2 - \mathcal{M}5$	Fibrosis, located_In, Pancreas, has_Origin, Genetic_Origin	$\mathcal{M}2$	No
3	$\mathcal{M}1, \mathcal{M}2$	$\mathcal{M}3 - \mathcal{M}5$	Genetic_Fibrosis	$\mathcal{M}3$	No
4	$\mathcal{M}1 - \mathcal{M}3$	$\mathcal{M}4, \mathcal{M}5$	–	$\mathcal{M}4$	No
5	$\mathcal{M}1 - \mathcal{M}4$	$\mathcal{M}5$	–	$\mathcal{M}5$	Yes
6	$\mathcal{M}1 - \mathcal{M}4$	–	–	–	

Proposition 8 (Correctness of Algorithm 1)
For every input $\mathcal{Q}$ and $\mathbf{S}$, Algorithm 1 computes a locality-based $\mathbf{S}$-module in $\mathcal{Q}$.

Proof. We have to show that (1) Algorithm 1 terminates for every input, and (2) the output extract_module($\mathbf{S}$, $\mathcal{Q}$) is a locality-based $\mathbf{S}$-module in $\mathcal{Q}$.

(1) Termination of the algorithm follows from the fact that in every iteration of the while loop either the size of $\mathcal{Q}_1$ increases, or the size of $\mathcal{Q}_1$ remains the same but the size of $\mathcal{Q}_2$ decreases. Note that this means that Algorithm 1 terminates in quadratic time in the number of axioms in $\mathcal{Q}$, assuming constant time locality test.

(2) It is easy to observe that every axiom α that is neither in $\mathcal{Q}_1$ nor in $\mathcal{Q}_2$ is local w.r.t. $\mathbf{S} \cup \mathsf{Sig}(\mathcal{Q}_1)$, since the only way such an α can appear is at the line 3 of the algorithm, and α remains in $\mathcal{Q} \setminus (\mathcal{Q}_1 \cup \mathcal{Q}_2)$ only if $\mathbf{S} \cup \mathsf{Sig}(\mathcal{Q}_1)$ does not change. $\diamond$

Note that there is an implicit non-determinism in Algorithm 1, namely, in line 3 in which an axiom from $\mathcal{Q}_2$ is selected. It might well be the case that several choices for α are possible at this moment. For example, the trace in Table 6.3 makes a different choice for α from $\mathcal{Q}_2$ than the trace in Table 6.2. In the first iteration of the while loop, we select $\alpha = \mathcal{M}2$ from $\mathcal{Q}_2$ instead of $\mathcal{M}1$ as in Table 6.2. This has resulted in a longer trace yet with the same result $\mathcal{Q}_1 = \{\mathcal{M}1, \ldots, \mathcal{M}4\}$. Note that axioms $\mathcal{M}2$ and $\mathcal{M}3$ are selected several times and produce different results for the locality tests, since $\mathcal{Q}_1$ has been modified. This demonstrates the reason why we reset $\mathcal{Q}_2$ to $\mathcal{Q} \setminus \mathcal{Q}_2$ at the line 8 of Algorithm 1, namely, not to miss axioms that has been checked to be local w.r.t. old $\mathcal{Q}_1$, but are no longer local w.r.t. new $\mathcal{Q}_1$.

As we have seen from the traces in Table 6.2 and Table 6.3, Algorithm 1 has produced the same output despite the fact that different choices for α has been made inside the while loop. One might wonder if this is always the case. It turns out that the choices for α indeed do not have any impact on the result of Algorithm 1, provided that the locality test satisfy some rather natural requirements:

Definition 7. *We say that a locality test locality_test$(\alpha, \mathbf{S})$ is* anti-monotonic *if for every $\mathbf{S}_1 \subseteq \mathbf{S}_2$, whenever locality_test$(\alpha, \mathbf{S}_2)$ succeeds then locality_test$(\alpha, \mathbf{S}_1)$ succeeds as well.*

Table 6.3. An alternative trace of Algorithm 1 for the input $\mathcal{Q} = \{\mathcal{M}1, \ldots, \mathcal{M}5\}$ and $\mathbf{S} = \{\mathsf{Cystic\_Fibrosis},\ \mathsf{Genetic\_Disorder}\}$

	$\mathcal{Q}_1$	$\mathcal{Q}_2$	New elements in $\mathbf{S} \cup \mathsf{Sig}(\mathcal{Q}_1)$	α	loc.
1	–	$\mathcal{M}1 - \mathcal{M}5$	Cystic_Fibrosis, Genetic_Disorder	$\mathcal{M}2$	Yes
2	–	$\mathcal{M}1, \mathcal{M}3 - \mathcal{M}5$	–	$\mathcal{M}3$	Yes
3	–	$\mathcal{M}1, \mathcal{M}4, \mathcal{M}5$	–	$\mathcal{M}1$	No
4	$\mathcal{M}1$	$\mathcal{M}2 - \mathcal{M}5$	Fibrosis, located_In, Pancreas, has_Origin, Genetic_Origin	$\mathcal{M}3$	No
5	$\mathcal{M}1, \mathcal{M}3$	$\mathcal{M}2, \mathcal{M}4, \mathcal{M}5$	Genetic_Fibrosis	$\mathcal{M}4$	No
6	$\mathcal{M}1, \mathcal{M}3, \mathcal{M}4$	$\mathcal{M}2, \mathcal{M}5$	–	$\mathcal{M}5$	Yes
7	$\mathcal{M}1, \mathcal{M}3, \mathcal{M}4$	$\mathcal{M}2$	–	$\mathcal{M}2$	No
8	$\mathcal{M}1 - \mathcal{M}4$	$\mathcal{M}5$	–	$\mathcal{M}5$	Yes
9	$\mathcal{M}1 - \mathcal{M}4$	–	–	–	

We say that locality of $\mathcal{Q}_1$ w.r.t. $\mathbf{S}$ in $\mathcal{Q}_1$ is provable using locality_test$(\alpha, \mathbf{S})$ *if for every $\alpha \in \mathcal{Q} \setminus \mathcal{Q}_1$, we have that locality_test*$(\alpha, \mathbf{S} \cup \mathsf{Sig}(\mathbf{S}_1))$ *succeeds.* $\diamond$

Proposition 9 (Determinism of Algorithm 1)
The output of Algorithm 1 based on anti-monotonic locality_test$(\alpha, \mathbf{S})$ is the smallest $\mathcal{Q}_1$ such that locality of $\mathcal{Q}_1$ w.r.t. $\mathbf{S}$ is provable using locality_test$(\alpha, \mathbf{S})$.

Proof. It is easy to see (see the proof of Proposition 8) that the locality of every output $\mathcal{Q}_1$ of Algorithm 1 is provable using locality_test$(\alpha, \mathbf{S})$. It remains, thus, to show that for every subset $\mathcal{Q}_1' \subseteq \mathcal{Q}$ such that locality of $\mathcal{Q}_1'$ w.r.t. $\mathbf{S}$ in $\mathcal{Q}$ is provable using locality_test$(\alpha, \mathbf{S})$, we have $\mathcal{Q}_1 \subseteq \mathcal{Q}_1'$.

Assume, to the contrary, that for some run of the algorithm, the output $\mathcal{Q}_1$ is not a subset of $\mathcal{Q}_1'$. Since the initial $\mathcal{Q}_1 = \emptyset$ was a subset of $\mathcal{Q}_1'$, there is a moment in the computation such that $\mathcal{Q}_1$ was a subset of $\mathcal{Q}_1'$, but $\mathcal{Q}_1 \cup \{\alpha\}$ is no longer a subset of $\mathcal{Q}_1'$. For these particular values of $\mathcal{Q}_1$ and α we have: (i) $\mathcal{Q}_1 \subseteq \mathcal{Q}_1'$, (ii) $\alpha \in \mathcal{Q} \setminus \mathcal{Q}_1'$, and (iii) locality_test$(\alpha, \mathbf{S} \cup \mathsf{Sig}(\mathcal{Q}_1))$ fails. From (ii) by property of $\mathcal{Q}_1'$ we have locality_test$(\alpha, \mathbf{S} \cup \mathsf{Sig}(\mathcal{Q}_1'))$ succeeds, which implies using (i) and anti-monotonicity of locality_test that locality_test$(\alpha, \mathbf{S} \cup \mathsf{Sig}(\mathcal{Q}_1))$ succeeds which contradicts to (iii). This proves that $\mathcal{Q}_1$ is indeed a subset of $\mathcal{Q}_1'$. $\diamond$

Corollary 5 (Uniqueness of a Minimal Locality-Based Module)
Algorithm 1 using a test based on the semantic locality produces a unique minimal locality-based $\mathbf{S}$-module in $\mathcal{Q}$.

Proof. By Lemma 2 the semantic locality admits anti-monotonicity. $\diamond$

Corollary 6 (Uniqueness of a Minimal Syntactic Locality-Based Module)
Algorithm 1 using a test based on the syntactic locality produces a unique minimal syntactic locality-based $\mathbf{S}$-module in $\mathcal{Q}$.

Proof. By Lemma 3 the syntactic locality admits anti-monotonicity. $\diamond$

6.5.4 Properties of Locality-Based Modules

In this section, we outline some interesting properties of locality-based modules which make it possible to use them for applications other than knowledge reuse.

Let $\mathcal{Q}^{loc}_{\mathbf{S}}$ be the smallest locality-based $\mathbf{S}$-module in $\mathcal{Q}$, which is unique by Corollary 5 and is the output of Algorithm 1 for $\mathcal{Q}$ and $\mathbf{S}$. The first property is a direct consequence of Corollary 5:

Proposition 10. *$\mathcal{Q}^{loc}_{\mathbf{S}}$ contains all $\mathbf{S}$-essential axioms in $\mathcal{Q}$ w.r.t. every language* L *with Tarski-style set-theoretic semantics.*

Proof. Let $\mathcal{Q}_1$ be a minimal $\mathbf{S}$-module in $\mathcal{Q}$. We need to show that $\mathcal{Q}_1 \subseteq \mathcal{Q}^{loc}_{\mathbf{S}}$. Since (i) $\mathcal{Q}_1$ is a subset of a locality-based $\mathbf{S}$-module in $\mathcal{Q}$ (say, of $\mathcal{Q}$ itself) and (ii) there is no proper subset of $\mathcal{Q}_1$ that is a locality-based $\mathbf{S}$-module in $\mathcal{Q}$, we have that $\mathcal{Q}_1$ is a subset of a minimal locality-based $\mathbf{S}$-module in $\mathcal{Q}$. Since such a module is unique by Corollary 5, and it is $\mathcal{Q}^{loc}_{\mathbf{S}}$, we have that $\mathcal{Q}_1 \subseteq \mathcal{Q}^{loc}_{\mathbf{S}}$. ◇

As shown in Table 6.2 and Table 6.3, the minimal locality-based $\mathbf{S}$-module extracted from $\mathcal{Q}$ contains all $\mathbf{S}$-essential axioms $\mathcal{M}1$–$\mathcal{M}4$. In our case, the module contains only essential axioms; in general, however, locality-based modules might contain non-essential axioms; otherwise, they would provide a solution for our task T3 in (6.4).

Proposition 11. *Let $\mathcal{Q}$ be ontology, A and B atomic concepts and $\mathbf{S}_{(i)}$ a signature. Then:*

1. $\mathbf{S}_1 \subseteq \mathbf{S}_2$ *implies* $\mathcal{Q}^{loc}_{\mathbf{S}_1} \subseteq \mathcal{Q}^{loc}_{\mathbf{S}_2}$ *(monotonicity);*
2. $\mathcal{Q} \models (A \sqsubseteq B)$ *iff* $\mathcal{Q}^{loc}_{\{A\}} \models (A \sqsubseteq B)$.
3. $\mathcal{Q} \models (A \sqsubseteq B)$ *implies* $\mathcal{Q}^{loc}_{\{B\}} \subseteq \mathcal{Q}^{loc}_{\{A\}}$ *or* $\mathcal{Q}^{loc}_{\{A\}} \models A \sqsubseteq \bot$.

Proof

1. Since $\mathcal{Q}^{loc}_{\mathbf{S}_2}$ is a locality-based $\mathbf{S}_2$-module in $\mathcal{Q}$, we have $\mathcal{Q} \setminus \mathcal{Q}^{loc}_{\mathbf{S}_2}$ is local w.r.t. $\mathbf{S}_2 \cup \mathsf{Sig}(\mathcal{Q}^{loc}_{\mathbf{S}_2})$. By anti-monotonicity of locality (see Lemma 2), $\mathcal{Q} \setminus \mathcal{Q}^{loc}_{\mathbf{S}_2}$ is local w.r.t. $\mathbf{S}_1 \cup \mathsf{Sig}(\mathcal{Q}^{loc}_{\mathbf{S}_2})$, hence $\mathcal{Q}^{loc}_{\mathbf{S}_2}$ is a locality-based $\mathbf{S}_1$-module in $\mathcal{Q}$. Since $\mathcal{Q}^{loc}_{\mathbf{S}_1}$ is contained in every locality-based $\mathbf{S}_1$-module in $\mathcal{Q}$ by Corollary 5, we have $\mathcal{Q}^{loc}_{\mathbf{S}_1} \subseteq \mathcal{Q}^{loc}_{\mathbf{S}_2}$.

2. The "if" part of this property is trivial since $\mathcal{Q}^{loc}_{\{A\}} \subseteq \mathcal{Q}$. In order to prove the "only if" part of the property, assume that $\mathcal{Q} \models (A \sqsubseteq B)$. Let $\mathbf{S} := \mathsf{Sig}(\mathcal{Q}^{loc}_{\{A\}}) \cup \{A\}$, and consider the following two cases:

(a) $B \in \mathbf{S}$. Then by Remark 1, $\mathcal{Q}^{loc}_{\{A\}}$ is an $\mathbf{S}$-module in $\mathcal{Q}$, and so, $\mathcal{Q}^{loc}_{\{A\}} \models (A \sqsubseteq B)$ since $\mathsf{Sig}(A \sqsubseteq B) \subseteq \mathbf{S}$.

(b) $B \notin \mathbf{S}$. We demonstrate that $\mathcal{Q}^{loc}_{\{A\}} \models A \sqsubseteq \bot$ which suffices for proving $\mathcal{Q}^{loc}_{\{A\}} \models A \sqsubseteq B$.

Assume, to the contrary, that $\mathcal{Q}^{loc}_{\{A\}} \not\models A \sqsubseteq \bot$. Then there exists an $\mathbf{S}$-interpretation $\mathcal{I}$ such that $\mathcal{I} \models \mathcal{Q}^{loc}_{\{A\}}$ and $A^{\mathcal{I}} \neq \emptyset$. Let $\mathcal{I}'$ be a trivial expansion of

$\mathcal{I}$ to $\mathbf{S} \cup \mathsf{Sig}(\mathcal{Q})$. Since $\mathcal{Q}^{loc}_{\{A\}}$ is a locality-based $\mathbf{S}$-module in $\mathcal{Q}$ (see Remark 1 and Remark 2), we have $\mathcal{I}' \models \mathcal{Q}$. However, $\mathcal{I}'$ is not a model of $(A \sqsubseteq B)$ since $A^{\mathcal{I}'} \neq \emptyset$, but $B^{\mathcal{I}'} = \emptyset$ since $B \notin \mathbf{S}$. This contradicts to the assumption $\mathcal{Q} \models A \sqsubseteq B$.

3. As has been shown in the proof of property 2 above, if $\mathcal{Q} \models (A \sqsubseteq B)$, then either $B \in \mathsf{Sig}(\mathcal{Q}^{loc}_{\{A\}})$ or $\mathcal{Q}^{loc}_{\{A\}} \models A \sqsubseteq \bot$. So, it remains to show that $B \in \mathsf{Sig}(\mathcal{Q}^{loc}_{\{A\}})$ implies that $\mathcal{Q}^{loc}_{\{B\}} \subseteq \mathcal{Q}^{loc}_{\{A\}}$. Indeed, by Remark 1, $\mathcal{Q}^{loc}_{\{A\}}$ is a locality-based $(\mathsf{Sig}(\mathcal{Q}^{loc}_{\{A\}}) \cup \{A\})$-module in $\mathcal{Q}$. Since $B \in \mathsf{Sig}(\mathcal{Q}^{loc}_{\{A\}})$, then, in particular, $\mathcal{Q}^{loc}_{\{A\}}$ is a locality-based $\{B\}$-module in $\mathcal{Q}$. Since $\mathcal{Q}^{loc}_{\{B\}}$ is contained in every locality-based $\{B\}$-module in $\mathcal{Q}$, we have $\mathcal{Q}^{loc}_{\{B\}} \subseteq \mathcal{Q}^{loc}_{\{A\}}$ what was required to prove. $\diamond$

Proposition 11 gives two interesting properties of locality-based modules. The first one states that such modules may only grow if the input signature extends. The second one implies that the module for a single atomic concept A provides complete information about all the super-classes of A. This property can be used for optimizing classification: in order to *classify an ontology* $\mathcal{Q}$, i.e. to compute all *subsumption relations* $A \sqsubseteq B$ between pairs A, B of atomic concepts in $\mathcal{Q}$, it is sufficient to (1) extract all modules $\mathcal{Q}^{loc}_{\{A\}}$ of $\mathcal{Q}$ for each atomic concept A (2) classify each of these modules *independently* (possibly *in parallel*), and (3) merge the results of the individual classifications. By Property 2, if the subsumption $A \sqsubseteq B$ is implied by the ontology $\mathcal{Q}$ then it is implied by the module $\mathcal{Q}^{loc}_{\{A\}}$ and, hence, it will be obtained in step (2).

Finally, Property 3 in Proposition 10 can also be used to optimize classification. The property provides a necessary condition for a subsumption $A \sqsubseteq B$ to hold in an ontology, which can be used to quickly detect *non-subsumptions*: If the inclusion $\mathcal{Q}^{loc}_{\{B\}} \subseteq \mathcal{Q}^{loc}_{\{A\}}$ between the minimal locality-based modules does not hold, and A is found to be satisfiable, then a reasoner does not need to prove the subsumption $A \sqsubseteq B$ w.r.t. $\mathcal{Q}$, since it never holds.

We have used these properties of locality-based modules for optimizing incremental classification—that is, to improve the efficiency of DL reasoners under changes in the ontology. We refer the interested reader to [8] for details.

6.6 Implementation and Evaluation

In this section, we show that minimal locality-based modules obtained from realistic ontologies are small enough to be useful in practice. For evaluation and comparison, we have implemented the following algorithms using Manchester's OWL API:[3]

A1:The PROMPT-FACTOR algorithm, as described in [15];
A2:The algorithm for extracting modules described in [6];
A3:Our algorithm for extracting modules (Algorithm 1), based on locality.

As a test suite, we have collected a set of well-known ontologies available on the Web, which can be divided into two groups:

[3] `http://sourceforge.net/projects/owlapi`

Simple. In this group, we have included the National Cancer Institute (NCI) Ontology,[4] the SUMO Upper Ontology,[5] and the Gene Ontology (GO),[6] These ontologies are expressed in a simple ontology language and are of a simple structure; in particular, they do not contain GCIs, but only definitions.

Complex. This group contains the well-known GALEN ontology (GALEN-Full),[7] the DOLCE upper ontology (DOLCE-Lite),[8] and NASA's Semantic Web for Earth and Environmental Terminology (SWEET)[9]. These ontologies are complex since they use many constructors from OWL DL and/or include a significant number of GCIs. In the case of GALEN, we have also considered a version GALEN-Small that has commonly been used as a benchmark for OWL reasoners. This ontology is almost 10 times smaller than the original GALEN-Full ontology, yet similar in structure.

Table 6.4. Comparison of Different Modularization Algorithms

Ontology	♯ Atomic	A1: Prompt-Factor		A2: Mod. in [6]		A3: Loc.-based mod.	
	Concepts	Max.(%)	Avg.(%)	Max.(%)	Avg.(%)	Max.(%)	Avg.(%)
NCI	27772	87.6	75.84	55	30.8	0.8	0.08
GO	22357	1	0.1	1	0.1	0.4	0.05
SUMO	869	100	100	100	100	2	0.09
GALEN-Small	2749	100	100	100	100	10	1.7
GALEN-Full	24089	100	100	100	100	29.8	3.5
SWEET	1816	96.4	88.7	83.3	51.5	1.9	0.1
DOLCE-Lite	499	100	100	100	100	37.3	24.6

For each of these ontologies, and for each atomic concept in their signature, we have extracted the corresponding modules using algorithms A1-A3 and measured their size. We use modules for single atomic concepts to get an idea of the typical size of locality-based modules compared to the size of the whole ontology. Also, as seen before, modules for atomic concepts are especially interesting for optimized classification of ontologies.

The results we have obtained are summarized in Table 6.4. The table provides the size of the largest module and the average size of the modules obtained using each of these algorithms. In the table, we can clearly see that locality-based modules are significantly smaller than the ones obtained using the other methods; in particular, in the case of SUMO, DOLCE, and GALEN the algorithms A1 and A2 retrieve the whole ontology as the module for each atomic concept. In contrast, the modules

[4] `http://www.mindswap.org/2003/CancerOntology/nciOncology.owl`
[5] `http://ontology.teknowledge.com/`
[6] `http://www.geneontology.org`
[7] `http://www.openclinical.org/prj_galen.html`
[8] `http://www.loa-cnr.it/DOLCE.html`
[9] `http://sweet.jpl.nasa.gov/ontology/`

we obtain using our algorithm are significantly smaller than the size of the input ontology. In fact, our modules are not only smaller, but are also strict subsets of the respective modules computed using A1 and A2.

For NCI, GO and SUMO, we have obtained very small locality-based modules. This can be explained by the fact that these ontologies, even if large, are simple in structure and logical expressivity. For GALEN, SWEET and DOLCE, the locality-based modules are larger. Indeed, the largest module in GALEN-Small is 1/10 of the size of the ontology. For DOLCE, the modules are even bigger—1/3 of the size of the ontology—which indicates that the dependencies between the different concepts in the ontology are very strong and complicated. The SWEET ontology is an exception: even though the ontology uses most of the constructors available in OWL, the ontology is heavily underspecified, which yields small modules.

6.7 Conclusion

We have proposed a definition of a module for a given vocabulary within an ontology to be reused. Based on this definition, we have formulated three reasoning problems concerning the extraction of minimal modules and shown that none of them is algorithmically solvable, even for simple fragments of OWL DL. We have introduced locality-based modules as an approximation to minimal modules and have empirically demonstrated that such modules are reasonably small for many real-world ontologies.

References

1. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope. In: Proc. IJCAI 2005, pp. 364–370 (2005)
2. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, Cambridge (2003)
3. Börger, E., Grädel, E., Gurevich, Y.: The Classical Decision Problem. In: Perspectives of Mathematical Logic. Springer, Heidelberg (1997); Second printing (Universitext) 2001
4. Cuenca Grau, B., Horrocks, I., Kazakov, Y., Sattler, U.: A logical framework for modularity of ontologies. In: Proc. IJCAI 2007, pp. 298–304 (2007)
5. Cuenca Grau, B., Horrocks, I., Kutz, O., Sattler, U.: Will my Ontologies Fit Together? In: Proc. DL 2006 (2006)
6. Cuenca Grau, B., Parsia, B., Sirin, E., Kalyanpur, A.: Modularity and Web Ontologies. In: Proc. KR 2006, pp. 198–209 (2006)
7. Ghilardi, S., Lutz, C., Wolter, F.: Did I Damage my Ontology? A Case for Conservative Extensions in Description Logics. In: Proc. KR 2006, pp. 187–197 (2006)
8. Cuenca Grau, B., Halaschek-Wiener, C., Kazakov, Y.: History matters: Incremental ontology reasoning using modules. In: Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) ASWC 2007 and ISWC 2007. LNCS, vol. 4825, pp. 183–196. Springer, Heidelberg (2007)

9. Horrocks, I., Patel-Schneider, P.F., van Harmelen, F.: From $\mathcal{SHIQ}$ and RDF to OWL: The making of a web ontology language. J. of Web Semantics 1(1), 7–26 (2003)
10. Horrocks, I., Sattler, U.: A tableaux decision procedure for SHOIQ. In: Proc. of the IJCAI. Morgan Kaufman, San Francisco (2005)
11. Kalyanpur, A., Parsia, B., Sirin, E., Cuenca Grau, B., Hendler, J.: SWOOP: A web editing browser. Elsevier's Journal Of Web Semantics 4(2), 144–153 (2006)
12. Lutz, C., Walther, D., Wolter, F.: Conservative extensions in expressive description logics. In: Proc. of IJCAI 2007, pp. 453–459 (2007)
13. Möller, R., Haarslev, V.: Description logic systems. In: The Description Logic Handbook, ch. 8, pp. 282–305. Cambridge University Press, Cambridge (2003)
14. Motik, B.: Reasoning in Description Logics using Resolution and Deductive Databases. PhD thesis, Univesität Karlsruhe (TH), Karlsruhe, Germany (2006)
15. Noy, N.F., Musen, M.A.: The PROMPT suite: Interactive tools for ontology mapping and merging. Int. Journal of Human-Computer Studies 6(59) (2003)
16. Patel-Schneider, P.F., Hayes, P., Horrocks, I.: Web ontology language OWL Abstract Syntax and Semantics. W3C Recommendation (2004)
17. Schmidt-Schauß, M., Smolka, G.: Attributive concept descriptions with complements. Artif. Intell. 48(1), 1–26 (1991)
18. Sirin, E., Parsia, B.: Pellet system description. In: Proc. DL 2004 (2004)
19. Tsarkov, D., Horrocks, I.: FaCT++ description logic reasoner: System description. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS, vol. 4130, pp. 292–297. Springer, Heidelberg (2006)

7

Structure-Based Partitioning of Large Ontologies

Heiner Stuckenschmidt and Anne Schlicht

Universität Mannheim, Germany
{heiner,anne}@informatik.uni-mannheim.de

Summary. In this chapter we describe a method for structure-based ontology partitioning and its implementation that is practically applicable to very large ontologies. We show that a modularization based on structural properties of the ontology only already results in modules that intuitively make sense. The method was used for creating an overview graph for ontologies and for extracting key topics from an ontology that correspond to topics selected by human experts. Because the optimal modularization of an ontology greatly depends on the application it is used for, we implemented the partitioning algorithm in a way that allows for adaption to different requirements. Furthermore this adaption can be performed automatically by specifying requirements of the application.

7.1 Introduction

In our work, we focus on the task of splitting up an existing ontology into a set of modules according to some criteria that define the notion of a good modularization. Intuitively, we can say that a module should contain information about a coherent subtopic that can stand for itself. This requires that the concepts within a module are semantically connected to each other and do not have strong dependencies with information outside the module. These considerations imply the need for a notion of *dependency* between concepts that needs to be taken into account. There are many different ways in which concepts can be related explicitly or implicitly. At this point we abstract from specific kinds of dependencies and choose a general notion of dependency between concepts. The resulting model is the one of a weighted graph $O = \langle C, D, w \rangle$ where nodes C represent concepts and links D between concepts represent different kinds of dependencies that can be weighted according to the strength of the dependency. These dependencies can reflect the definitions of the ontology or can be implied by the intuitive understanding of concepts and background knowledge about the respective domain. Looking for an automatic partitioning method, we are only interested in such kinds of dependencies that can be derived from the ontology itself. This leads us to a first central assumption underlying our approach:

> **Assumption 1:** Dependencies between concepts can be derived from the structure of the ontology.

Depending on the representation language, different structures can be used as indicators of dependencies. These structures can be subclass relations between classes, other relations linked to classes by the range and domain restrictions or the appearance of a class name in the definition of another class. In previous work, we have shown that this assumption is valid in many cases [13]. A second basic assumption of our approach that directly follows from the first assumption and will be the focus of this paper is the following:

> **Assumption 2:** The quality of a modularization can be determined on the basis of the structure of the individual modules and the connections between them.

This assumption does not only provide a rationale for structure-based ontology partitioning, it also allows us to adapt the partitioning algorithm originally proposed in [13] by explicitly taking structural criteria for measuring the quality of the resulting modular ontology into account.

In the subsequent section we describe the different steps of the partitioning algorithm. Section 3 comprises an overview of the implementation including instruction for its utilization. We demonstrate application of the tool for visualization and identification of key topics in section 4. The last section summarizes the main ideas and results and gives an outlook on future work.

7.2 Algorithm

Our algorithm consists of three tasks that are executed in six independent steps. The first task (Steps 1.1 and 1.2) is the creation of a dependency graph from an ontology definition. Guided by this graph the actual partitioning is the second task. The third task is optimization of the partitioning by assignment of isolated concepts, merging some modules and duplicating selected axioms. Step 4 describes how parameters that are required by the different partitioning steps are determined automatically, based on a given set of criteria.

7.2.1 Dependency Graph

The first task of the algorithm is the conversion of an ontology in OWL, RDF or KIF format into a weighted graph. It consists of two steps, the creation of the graph and the computation of the weights.

Step 1.1: *Create Dependency Graph:* In the first step a dependency graph is extracted from an ontology source file. The elements of the ontology (concepts, relations, instances) are represented by nodes in the graph. Links are introduced between nodes if the corresponding elements are related in the ontology. There are five types of relations between elements to choose from for the creation of links: subclass, property, definition, substring and distance relations.

When property relations are to be included, domain and range of each property are connected by a link. Definition relations are established between a concept and terms contained in its definition (either only properties or also other resources). These can be used to make concepts dependent on some shared property. The remaining two relations, substring and string distance, look at the concept names (or labels if specified). They create a relation if one concept name is contained in another or if the string distance between two concept names is below a certain threshold. The string relations are oportune when the terms of the ontology have a compositional structure. For example, [10] found that in the Gene ontology 65% of the terms encode a semantic relation in their name (e.g. *regulation of cell proliferation* is related to *cell proliferation*).

Step 1.2: *Determine Strength of Dependencies:* In the second step the strength of the dependencies between the concepts has to be determined. Following the basic assumption of our approach, we use the structure of the dependency graph to determine the weights of dependencies. In particular we use results from social network theory by computing the proportional strength network for the dependency graph. The strength of the dependency of a connection between a node c_i and c_j is determined to be the proportional strengths of the connection. The proportional strength describes the importance of a link from one node to the other based on the number of connections a node has. In general it is computed by dividing the sum of the weights of all connections between c_i and c_j by the sum of the weights of all connections c_i has to other nodes (compare [4], page 54ff):

$$w(c_i, c_j) = \frac{a_{ij} + a_{ji}}{\sum\limits_k a_{ik} + a_{ki}}$$

Here a_{ij} is the weight preassigned to the link between c_i and c_j - in the experiments reported in this section this will always be one. As a consequence, the proportional strength used in the experiments is one divided by the number of nodes c_i is connected to.

The intuition behind it is that individual social contacts become more important if there are only few of them. In our setting, this measure is useful because we want to prevent that classes that are only related to a low number of other classes get separated from them. This would be against the intuition that classes in a module should be related.

We use node **d** in Fig. 7.1 to illustrate the calculation of weights using the proportional strength. The node has four direct neighbors, this means that the proportional strength of the relation to these neighbors is 0.25 (one divided by four). Different levels of dependency between **d** and its neighbors now arise from the relative dependencies of the neighbors with **d** (the proportional strength is non-symmetric). We see

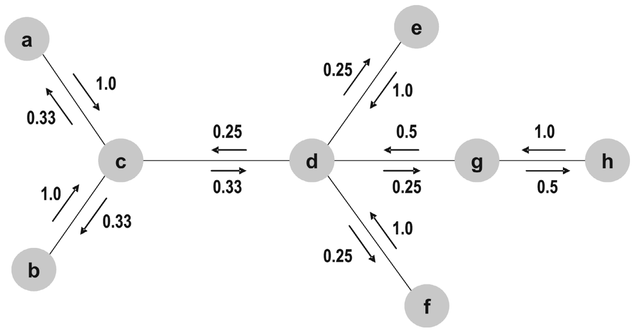


Fig. 7.1. An example graph with proportional strength dependencies

that **e** and **f** having no other neighbors completely depend on **d**. The corresponding value of the dependency is 1. Further, the strength of the dependency between **g** and **d** is 0.5, because **g** has two neighbors and the dependency between **b** and **d** is 0.33 as **b** has 3 neighbors.

7.2.2 Identification of Modules

Step 2: *Determine Modules* The proportional strength network provides us with a foundation for detecting sets of strongly related concepts. For this purpose, we make use of an algorithm that computes all maximal line islands of a given size in a graph [2].

Definition 1 (Line Island). *A set of vertices $I \subseteq C$ is a line island in a dependency graph $G = (C, D, w)$ if and only if*

- *I induces a connected subgraph of G*
- *There is a weighted graph $T = (V_T, E_T, w_T)$ such that:*
  - *T is embedded in G*
  - *T is an maximal spanning tree*[1] *with respect to I*
  - *the following equation holds:*

$$\max_{\{(v,v')\in D | (v\in I \wedge v' \notin I)\vee(v'\in I \wedge v \notin I)\}} w(v,v') < \min_{(u,u')\in E_T} w(u,u')$$

Note that for the determination of the maximal spanning tree the direction of edges is not considered. ◇

This criterion exactly coincides with our intuition about the nature of modules given in the introduction, because it determines sets of concepts that are stronger internally connected than to any other concept not in the set. The algorithm requires an upper and a lower bound on the size of the detected set as input and assigns an island

[1] A maximal spanning tree is a spanning tree with weight greater than or equal to the weight of every other spanning tree.

number to each node in the dependency graph. We denote the island number assigned to a concept c as $\alpha(c)$. The assignment $\alpha(c) = 0$ means that c could not be assigned to an island.

We use different sets of nodes in the graph in Fig. 7.1 to illustrate the concept of a line island. Let us first consider the set $\{\mathbf{a}, ..., \mathbf{f}\}$. It forms a connected subgraph. The maximal spanning tree of this set consists of the edges $\mathbf{a} \xrightarrow{1.0} \mathbf{c}$, $\mathbf{b} \xrightarrow{1.0} \mathbf{c}$, $\mathbf{c} \xrightarrow{0.33} \mathbf{d}$, $\mathbf{e} \xrightarrow{1.0} \mathbf{d}$, and $\mathbf{f} \xrightarrow{1.0} \mathbf{d}$. We can see however, that this node set is not an island, because the minimal weight of an edge in the spanning tree is 0.33 and there is an incoming edge with strength 0.5 ($\mathbf{g} \overset{0.5}{\rightarrow} \mathbf{d}$). If we look at the remaining set of nodes $\{\mathbf{g}, \mathbf{h}\}$, we see that it fulfills the conditions of an island: it forms a connected subgraph, the maximal spanning tree consists of the edge $\mathbf{h} \overset{1.0}{\rightarrow} \mathbf{g}$ and the maximal value of in- or outgoing links is 0.5 ($\mathbf{g} \overset{0.5}{\rightarrow} \mathbf{d}$). This set, however, is not what we are looking for because it is not maximal: it is included in the set $\{\mathbf{d}, ..., \mathbf{h}\}$. This set is a line island with the maximal spanning tree consisting of the edges $\mathbf{e} \xrightarrow{1.0} \mathbf{d}$, $\mathbf{f} \xrightarrow{1.0} \mathbf{d}$, $\mathbf{g} \xrightarrow{0.5} \mathbf{d}$ and $\mathbf{h} \xrightarrow{1.0} \mathbf{g}$ where the minimal weight (0.5) is higher than the maximal weight of any external link which is $\mathbf{c} \xrightarrow{0.33} \mathbf{d}$. Another reason for preferring this island is that the remaining node set $\{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$ also forms a line island with maximal spanning tree $\mathbf{a} \xrightarrow{1.0} \mathbf{c}$, $\mathbf{b} \xrightarrow{1.0} \mathbf{c}$ and the weaker external link $\mathbf{c} \xrightarrow{0.33} \mathbf{d}$.

The actual calculation of the islands is done by an external program written by Matjaz Zaversnik[2]. This program requires specification of minimum and maximum island sizes to compute the above defined line islands. The minimum size is always set to 1 for not restricting the island creation more than necessary.

7.2.3 Optimization 1: Assignment of Isolated Concepts

After partitioning, in some cases there will be some leftover nodes which are not assigned to any cluster. The algorithm will automatically assign these nodes based on the strength of the relations to nodes already assigned to a module.

Step 3.1: *Assign Isolated Concepts* Leftover nodes are assigned to the cluster to which they have the strongest connection. In particular this is the island of the neighboring node they have the strongest relation to. In cases where all neighboring nodes are unassigned as well, these nodes are assigned first.

How this process works can best be explained by an example. Fig. 7.2 shows an example network. It contains two modules (M1 and M2) and one leftover node (c8). c8 is connected to module M1 by one edge with strength 0.3 and to module M2 by two arcs with strengths 0.2 and 0.3. To determine the strength of a connection between a leftover node and a module, the strengths of all edges and arcs that connect the two are summed. Because edges are undirected and work in this respect in both directions, they can be considered twice as strong as arcs. Therefore their weight is doubled. In the example network the connection between c8 and M1 is 0.6, between c8 and M2 0.5, so the leftover node will be assigned to module M1.

[2] `http://vlado.fmf.uni-lj.si/pub/networks/`

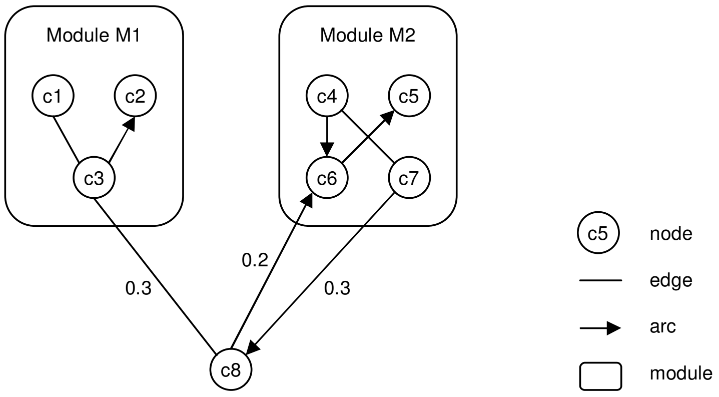


Fig. 7.2. Example network for the assignment of leftover nodes to modules

7.2.4 Optimization 2: Merging

Looking at the result of the example application we get a first idea about the strengths and weaknesses of the algorithm. We can see that the algorithm generates some modules that meet our intuition about the nature of a module quite well. In some cases subtrees that could be considered to form one module are further split even if the complete subtree does not exceed the upper size limit. This can be explained by an unbalanced modelling of the ontology as subtrees tend to be split up at concepts with a high number of direct subclasses compared to its sibling classes. This phenomenon often reflect a special importance of the respective concept in the ontology that also justifies the decision to create a separate model for this concept. The iterative strategy frees us from determining a lower bound for the size of modules. As a result, however, the algorithm sometimes create rather small modules. This normally happens when the root concept of a small subtree is linked to a concept that has many direct subclasses. For the result of the partitioning method these subsets are often pathological because a coherent topic is split up into a number of small modules that do not really constitute a sensible model on their own.

When inspecting the dependencies in the relevant parts of the hierarchy, we discovered that most of the problematic modules have very strong internal dependencies. In order to distinguish such cases, we need a measure for the strength of the internal dependency. The measure that we use is called the 'height' of an island. It uses the minimal spanning tree T used to identify the module: the overall strength of the internal dependency equals the strength of the weakest link in the spanning tree.

$$height(I) = \min_{(u,u') \in E_T} w(u,u')$$

We can again illustrate the the concept of height using the example from Fig. 7.1. We identified two islands, namely $\{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$ and $\{\mathbf{d}, ..., \mathbf{h}\}$. As the maximal spanning tree of the first island consists of the two edges $\mathbf{a} \xrightarrow{1.0} \mathbf{c}$, $\mathbf{b} \xrightarrow{1.0} \mathbf{c}$, the height of this

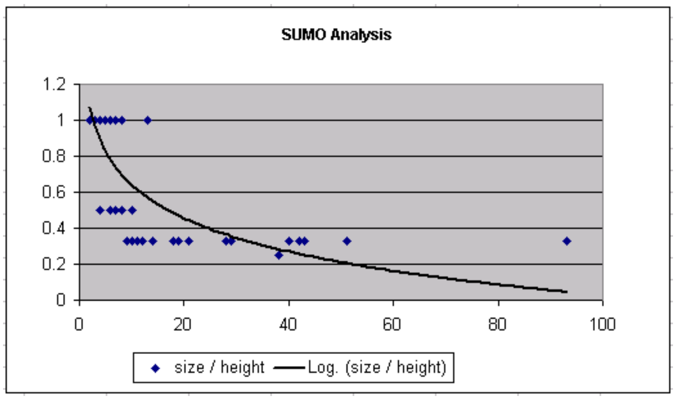


Fig. 7.3. Sizes and heights of partitions in the SUMO ontology

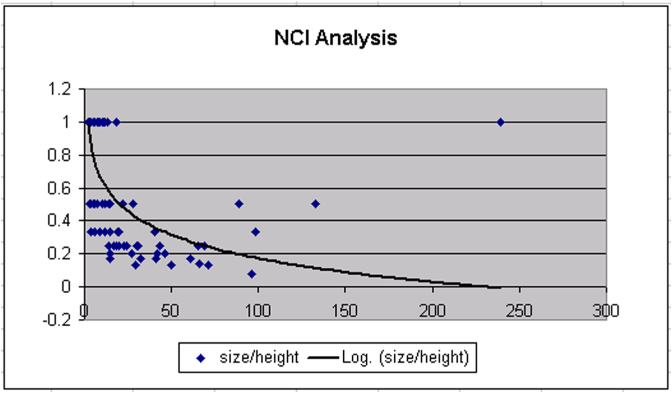


Fig. 7.4. Sizes and heights of partitions in the NCI ontology

island is 1.0. In the maximal spanning tree of the second island the edge $\mathbf{g} \overset{0.5}{\rightarrow} \mathbf{d}$ is the weakest link that therefore sets the height of the island to 0.5.

We found many cases where generated modules that do not make sense had an internal dependency of strength one. In a post-processing step this allows us to automatically detect critical modules. While for the case of an internal strength of one we almost never found the corresponding module useful in the context of the original ontology, it is not clear where to draw the line between a level of internal dependency that still defines sensible modules and a level that overrules important dependencies to concepts outside the module. In our experiments we made the experience that a threshold of 0.5 leads to good results in most cases.[3]

[3] Note that due to the calculation of the dependency value, the internal strength is always of the form $\frac{1}{n}$.

Figures 7.3 and 7.4 show the results of comparing the size and the height of computed islands. The plots clearly show a correlation between these properties. We also see that—except for one case—islands with a height of one are quite small.[4] The results of these experiments provided us with sufficient evidence that the height of an island is a useful criterion for judging the quality of a module. As described above, one of the findings in the first experiments was the strong correlation between size of modules and the degree of internal dependency. Further, we found out that small modules were unnatural in most cases. In a second experiment, we show that this result can be used to 'repair' the result of the straightforward partitioning.

Step 3.2: *Merging* All modules whose height reaches the given threshold are merged into adjacent modules with a lower height. In many cases, there is only one adjacent module to merge with. In cases where more than one adjacent module exist, the strength of the dependencies between the modules is used to determine the candidate for merging.

7.2.5 Optimization 3: Criteria Maximization

Starting point of a partitioning problem is usually an application in need for a partitioning that meets certain requirements. Investigation of applications for ontology modularization reveals that the criteria for determining a "good" partitioning depend heavily on the concrete application [6]. For enabling adjustment to different application requirements the parameters that influence the final partitioning are customizable.

In order to support users to chose the right setting for a given application, we do not force to directly provide values for the parameters mentioned above. Instead, the user is asked to select and rank quality criteria for the resulting partitioning. This frees the user from the need to understand the partitioning method and the influence of the different parameters. In contrast, the user only has to be concerned with the requirements of the application at hand. We believe that this is a big step towards enabling domain experts with limited knowledge about the technical details of representation languages for ontologies to use this technology.

It can be assumed that different applications may not only impose different weights for the built-in criteria but also require consideration of new criteria that are defined by the user. For usage of additional criteria our tool provides a simple interface. We assume that althought there may be various types of requirements, they all can be described by concrete measurements that map partitionings to decimal numbers. Different applications may share some measurements but disagree on their relative importance, it could even happen that one measurement is positive for one application and negative for another.

Criteria

The most obvious criteria used for optimization are *number of modules*, *avarage module size* and *variance of size*. Uniformity of the size distribution, is measured

[4] The exception is a part of the NCI ontology that lists all countries of the world and therefore contains 1 class with more than 200 subclasses.

by the criteria *bulkyness* and *granularity*. The intention of bulkyness is to indicate that some modules are to large, e.g. the largest module has almost the same size as the whole ontology. For obtaining a smooth function module sizes $n_i = |M_i|$ are mapped to values in $[0, 1]$ depending on the size n of the ontology.

$$bulkyness(n_i, n) = \frac{1}{2} - \frac{1}{2}\, cos(\pi \cdot \frac{n_i}{n})$$

These values are averaged over all modules (weighted by module sizes) to obtain a measurement for the whole partitioning. Similarly, granularity indicates that modules are to small, e.g. when half of the concepts are contained in modules of size 1.

$$granularity(n_i, n) = (\frac{1}{2} + \frac{1}{2}\, cos(\pi \cdot \frac{n_i}{n}))^{20}$$

The precise value of the exponent does not matter, 20 is a value that worked well in practice. The graphs depicted in Fig. 7.5 illustrates the choice of functions. If there are two modules of the same size the bulkyness value is 0.5. If one of these modules is further partitioned into modules with size 1% of the size of the ontology, the granularity value is 0.5. In addition to criteria computed from size and number of modules, *connectedness* depends on the links between the modules. In many applications the number of symbols shared between axioms in different modules should be as small as possible. We consider the fraction of inter-modules edges with respect to the total number of edges. A detailed investigation of this measurement can be found in [12].

$$connectedness = \frac{\#\{(v, v') \in D \mid \alpha(v) \neq \alpha(v')\}}{\#\{(v, v') \in D\}}$$

where D is the set of edges of the dependency graph and α the assignment to modules (see Def. 1).

For optimization depending on specific terms contained in the ontology two additional criteria are defined. *Number of relevant modules* and *relative size of relevant*

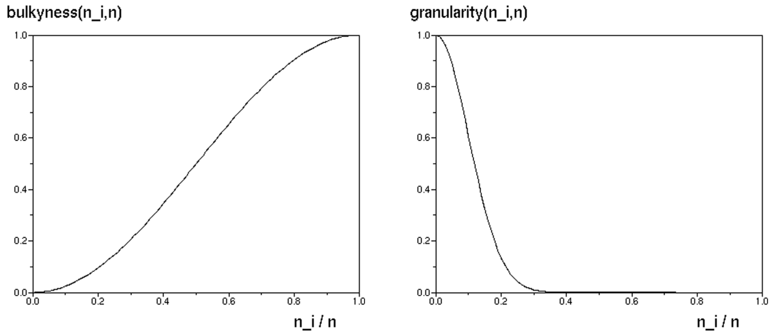


Fig. 7.5. Bulkyness and granularity of modules against the relative size

modules are computed depending on a given set of terms, i.e. a module is considered relevant if it contains one of the terms. For the relative size, the sum of the sizes of the relavant modules is compared to the size of the ontology.

There are different mechanisms for optimizing the configuration according to given criteria:

Dynamic defaults

Firstly, parameters that are not given in the settings file are determined automatically depending on given criteria. If for example the parameter "maximum island size" is not set in the input, it is set to $\frac{number\ of\ terms}{10} \cdot \frac{connectedness\ weight}{bulkyness\ weight}$. These dynamic default settings constitute an approximation of the optimal configuration.

Axiom Duplication

The preceding partitioning and optimization steps result in a non-redundant distributed representation of the source ontology. A term can not be allocated to more than one module. Sometimes it can be beneficial, however, to include certain axioms in several modules to decrease the connectedness of the resulting modularization. As on the other hand, copying axioms increases the redundancy, there has to be a tradeoff between these two conflicting requirements. Currently, this can be specified by setting an upper bound for the acceptable redundancy introduced. This means that the maximal redundancy is another parameter that influences the quality of the resulting partitioning in terms of connectedness and redundancy, Algorithm 2 shows the method for duplicating axioms based on a maximal redundancy value.[5]

Algorithm 2. Axiom Duplication

```
Require: partitioning: Set<Set<Axiom>>
Require: maxRedundancy: double
  limit = $\infty$
  candidates = $\emptyset$
  while redundancy < maxRedundancy & limit > 0 do
      for all (axiom,module) $\in$ partitioning do
          if numberOfLinks(axiom,module) > limit then
              candidates.add(axiom,module)
          end if
      end for
      for all (axiom,module) $\in$ candidates do
          if duplicating axiom to module decreases connectedness then
              duplicate(axiom, module, partitioning)
          end if
      end for
      limit = limit - 1
  end while
```

[5] A partitioning is represented by a set of modules which in turn are represented by sets of axioms.

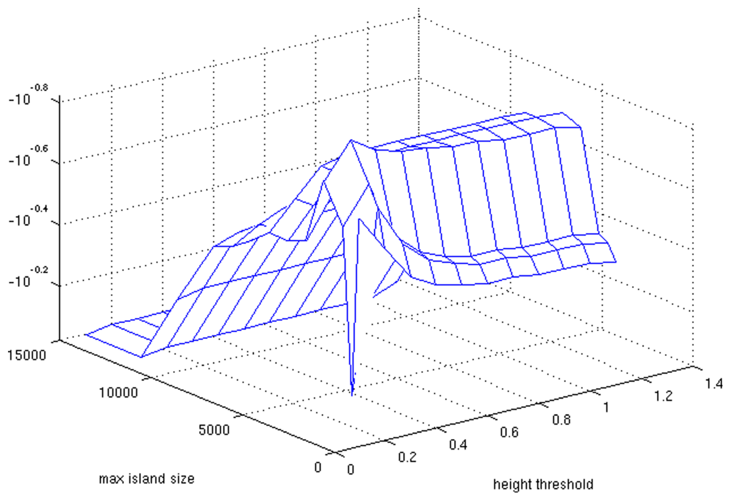


Fig. 7.6. Criteria-based determination of the configuration for the extraction application (see Sect. 7.4.3), displaying the value of $(-connectedness - 5 \cdot bulkyness)$

Step 3.3: *Axiom Duplication* Axioms with a high number of links to another module are copied to that module if connectedness is decreased by this duplication. Duplication stops when the maximum redundancy is reached.

Automatic configuration

The most advanced feature of the algorithm is the ability to automatically determine an optimal configuration of parameter settings:

Step 4: *Criteria-Based Optimization* Based on a set C of criteria and their weights w_c a configuration p is chosen that maximizes the weighted sum of the criteria values $v_{c,p}$.

$$\max_{p \in Config} \sum_{c \in C} w_c \cdot v_{c,p}$$

Figure 7.6 demonstrates the selection of the configuration. The highest point of the surface corresponds to the best configuration for the given criteria.

7.3 Tool Support for Automatic Partitioning

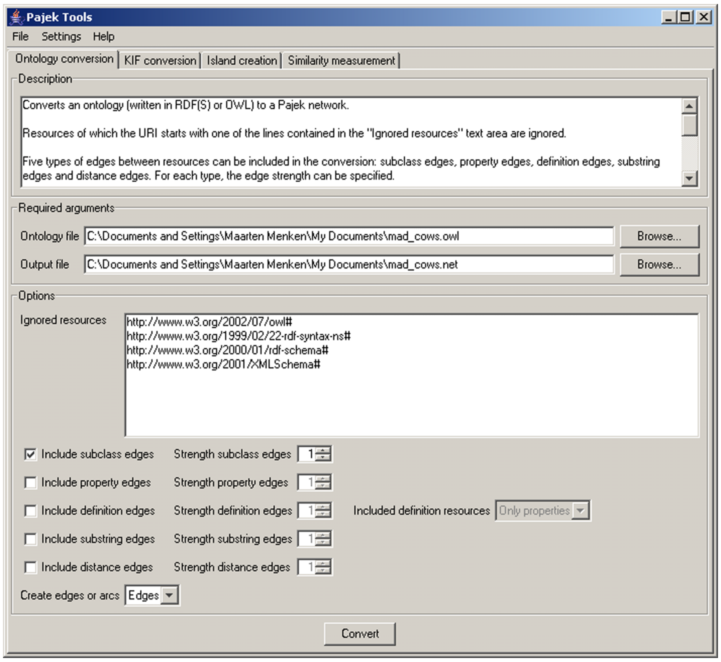
The algorithm described in the last sections is implemented in the *Pa*rtitioning *To*ol Pato, a Java application with two interfaces. Firstly, it performs the partitioning interactively through a graphical user interface. Secondly, the configuration can be specified in the settings file directly, providing control over additional parameters. A default configuration is computed, if only the source ontology file is specified. The tool is freely downloadable[6] and licensed under the GNU General Public

[6] `http://webrum.uni-mannheim.de/math/lski/Modularization/`

License. The features that where added in Pato1.3 (mainly criteria-based optimization and OWL-output) are not yet supported by the GUI, they are controlled using the settings file.

7.3.1 Graph Generation

Figure 7.7 shows a screen shot of the tool in which an OWL ontology is converted to a dependency network. The screen is divided into three parts: the upper part gives a short help text about the currently selected tab, the middle part is for specifying the required arguments (in this case the input ontology and the output network) and the bottom part is for various optional parameters that influence the conversion. The tool converts an ontology written in RDFS or OWL to a dependency graph, written in Pajek format. Once the ontology is converted to a Pajek network, it can be visualized and further processed using Pajek, a network analysis program.[7]



Converts an ontology (written in RDF(S) or OWL) to a Pajek network.

Resources of which the URI starts with one of the lines contained in the "Ignored resources" text area are ignored.

Five types of edges between resources can be included in the conversion: subclass edges, property edges, definition edges, substring edges and distance edges. For each type, the edge strength can be specified.

Ontology file: C:\Documents and Settings\Maarten Menken\My Documents\mad_cows.owl

Output file: C:\Documents and Settings\Maarten Menken\My Documents\mad_cows.net

Ignored resources:
http://www.w3.org/2002/07/owl#
http://www.w3.org/1999/02/22-rdf-syntax-ns#
http://www.w3.org/2000/01/rdf-schema#
http://www.w3.org/2001/XMLSchema#

[✓] Include subclass edges	Strength subclass edges: 1	
[ ] Include property edges	Strength property edges: 1	
[ ] Include definition edges	Strength definition edges: 1	Included definition resources: Only properties
[ ] Include substring edges	Strength substring edges: 1	
[ ] Include distance edges	Strength distance edges: 1	

Create edges or arcs: Edges

Fig. 7.7. Screen shot of the partitioning tool with the ontology conversion tab active

This tool uses Sesame, a system for storing and querying data in RDF and RDFS.[3] The ontology is loaded into a local Sesame repository, after which it can be easily queried via an API. Because Sesame does not have native OWL support, some extra programming had to be done to deal with ontologies in this format. This

[7] http://vlado.fmf.uni-lj.si/pub/networks/pajek/

includes explicitly querying for resources of type `owl:Class` while retrieving all classes (Sesame only returns resources of type `rdfs:Class`) and following blank nodes for determining the definition relations (see below).

To filter out irrelevant resources, the user can specify a number of namespaces that are to be ignored. They are entered in a text area (see Fig. 7.7). Resources that occur in those namespaces do not show up in the resulting network. In most cases the classes and properties defined in RDFS and OWL can be ignored, and by entering the corresponding namespaces in the text area those resources are prevented from appearing in the network.

Before converting an ontology, the user has to decide what relations to include in the network, and if those relations are to be represented by edges (undirected) or arcs (directed). The tool allows five types of relations to be included: subclass, property, definition, substring, and string distance relations. The user also has to decide about the strength of each type. At the moment, only subclass relations that are explicitly stated in the ontology are included in the network. No reasoners are used to infer new subclass relations. This will be added in a future version.

A simple but effective feature added in Pato 1.3 is the option to select if the values of “rdfs:label” or “rdf:ID” are prefered for vertex labels. Now the use of “rdfs:label” can be turned of by setting “ontology conversion - use labels=false” for ontologies that use labels for verbose descriptions or other purposes.

7.3.2 Partition Generation and Improvement

The actual creation of the partitions is based on the previously generated dependency network. It iteratively splits the network into smaller parts until a specified maximum number of concepts per cluster is reached. Alternatively the occurence of very large and very small clusters is measured and the iteration stops at the weighted optimum of these measures.

The actual calculation of the islands is done by an external Windows program written by Matjaz Zaversnik[8]. On Unix Systems Pato searches for Wine[9] and tries to use this application for executing the Windows program.

7.3.3 Criteria-Based Optimization

The criteria-based optimization can be performed using build in analysis methods and/or additional criteria. Currently Pato computes connectedness of modules and some measures that depend on the size distribution of the modules. Either the partitioned graph structure and the final resulting distributed ontology are subject to analysis.

The relevant criteria are specified in the settings file, with weights indicating their relative importance. For example “criteria weight - connectedness=3.7” sets the importance of the connectedness-criterion.

[8] `http://vlado.fmf.uni-lj.si/pub/networks/`

[9] `http://www.winehq.org/`

It can be assumed that different applications may not only impose different weights for the build-in criteria but also require consideration of new criteria that are defined by the user. For usage of additional criteria the simple interface "Analyse" is provided that contains two methods, one for setting the input and one for getting the result, both represented as instances of java.util.Properties. The new analyse-class computes the value of one or more new criteria, its "getResult"-methode then returns the criteria name - criteria value pairs as an Properties-instance. The only thing to do apart from implementing the computation of the new criteria is declaring the name of the new class in the settings file (e.g. "analyse class=some.package.name. AnalyseImplementation"). After registration the new class is used automatically without recompiling Pato.

7.3.4 Visualization of Criteria Dependencies

If lists of parameters are specified for the parameters "maximum island size" and/or "height threshold", the corresponding criteria values are additionally stored as matrices. The produced file can be loaded into matlab or scilab for plotting the dependencies between criteria and parameters, Fig. 7.6 was created this way. Furthermore the matrices can be used for efficient determination of the pareto-optimal[10] configuration. Especially when relative importance of criteria is vague it might be necessary to try different weights, the optimal configurations for all possible weight assignments are the pareto-optimal configurations.

7.3.5 Module Graph Creation

For creating the ontologies overviews like displayed in Fig. 7.8 Pajek is used. Pato generates a network file (named "...net") for the graph and a corresponding vector file ("...vec") that defines the vertex sizes. The two files are loaded via Pajeks graphical user interface. Drawing is initiated by selecting "Draw-Vector" from the Draw-menu. For determination of vertex labels Pajeks centrality calculation is performed on the dependency graph created by Pato. The resulting vector file is in the settings file as value of "centrality vector file" prior to the module graph creation.

7.3.6 Comparison and Evaluation

To evaluate the partitioning against some golden standard or against some other partitioning, the tool can calculate three similarity measurements: precision, recall and EdgeSim[9]. The first two measures are based on the numbers of intra-pairs, which are pairs of concepts that are in the same cluster[1]. The EdgeSim measure considers both the vertices and the edges and is not sensitive to the size and number of clusters (as are precision and recall). An intra-edge connects an intra-pair of vertices while an inter-edge connects vertices from different clusters.

[10] A pareto-optimal configuration is a configuration that can not be improved for any criterion without degrading the value of another criterion.

Precision: The precision of a partitioning is defined as the ratio of intra-pairs in the generated partitioning that are also intra-pairs in the optimal partitioning.

Recall: The recall of a partitioning is defined by the ratio of intra-pairs in the optimal partitioning that are also intra-pair in the generated one.

EdgeSim: The EdgeSim measure is defined by the ratio of edges that are either intra-edges in both partitions or inter-edges in both partitions.

The three measures give an indication of how well the partitioning was performed and therefore what relations and strengths give best results.

7.4 Application

The main application area for Pato is visualization and identification of the key topics of an ontology. Visualization devides into the different tasks of visualizing a whole ontology by identifying modules and partitioning for visualization of single modules. Considering single modules is also relevant for facilitated reasoning and is related to module extraction.

7.4.1 Visualization of Large Ontologies

Module Graph

Apart from the resulting OWL-modules, Pato generates networks that can be visualized using Pajek[11], a tool for large network analysis. The network shown in Fig. 7.8

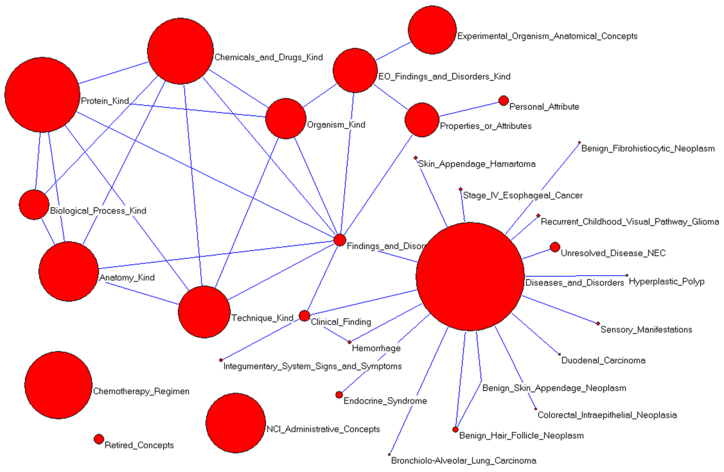

Experimental_Organism_Anatomical_Concepts
Chemicals_and_Drugs_Kind
EO_Findings_and_Disorders_Kind
Protein_Kind
Personal_Attribute
Organism_Kind
Properties_or_Attributes
Benign_Fibrohistiocytic_Neoplasm
Skin_Appendage_Hamartoma
Stage_IV_Esophageal_Cancer
Biological_Process_Kind
Recurrent_Childhood_Visual_Pathway_Glioma
Findings_and_Disor
Unresolved_Disease_NEC
Anatomy_Kind
Diseases_and_Disorders
Hyperplastic_Polyp
Technique_Kind
Clinical_Finding
Sensory_Manifestations
Hemorrhage
Integumentary_System_Signs_and_Symptoms
Duodenal_Carcinoma
Chemotherapy_Regimen
Benign_Skin_Appendage_Neoplasm
Endocrine_Syndrome
Colorectal_Intraepithelial_Neoplasia
NCI_Administrative_Concepts
Benign_Hair_Follicle_Neoplasm
Retired_Concepts
Bronchiolo-Alveolar_Lung_Carcinoma

Fig. 7.8. The module graphs displays the connections between modules. For each module the name of the vertex with the highest centrality labels the module.

[11] `http://vlado.fmf.uni-lj.si/pub/networks/pajek/`

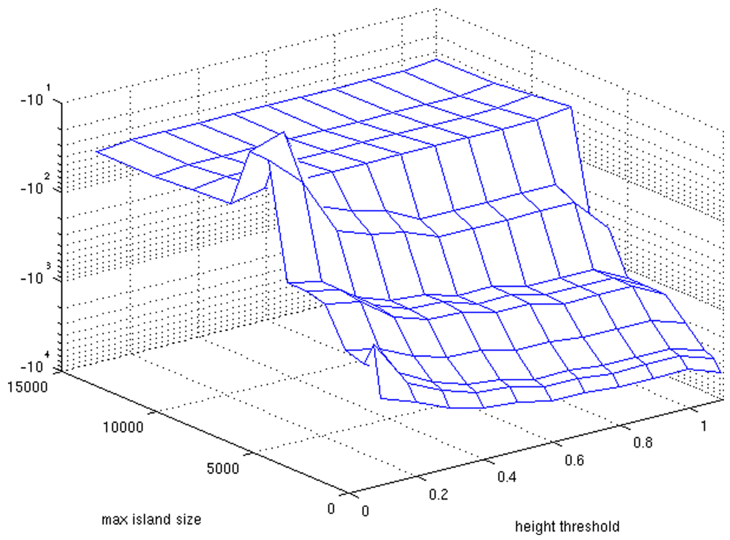


Fig. 7.9. The criteria evaluation for visualization displays the value of $-2 \cdot abs(numberOfModules - 30) - connectedness$

displays each module as a vertex, the size corresponding to the number of terms in the module. In addition to visualization, we used Pajek for determining the module labels. In particular, a module is labeled by the vertex with the highest *betweenness*[12], a centrality measurement defined by [8] for social networks.

For successful visualization of the whole ontology, the number of modules should be about 30 to provide as much information as can be displayed. Furthermore very large modules should be avoided. Therefore the criteria weights are set to (-1) for *connectedness* and (-2) for *abs(numberOfModules-30)*.

According to this criteria, Pato chooses the configuration[13]. Figure 7.9 shows the weighted sum of the criteria values. Dependency weights[14] where set directly, they depend on the type of relations that are to be visualized in the module graph.

7.4.2 Identification of Key Topics

We consider an imaginary optimal partitioning of the ontology. An automatically generated partitioning is evaluated against this optimal partitioning in terms of recall and precision.

[12] The betweenness of a vertex $v \in V$ is the percentage of shortest paths this vertex lies on: $betweenness(v) = \sum_{\substack{s,t \in V \\ s \neq v \neq t}} \frac{\sigma_{st}(v)}{\sigma_{st}}$ were σ_{st} is the number of shortest paths between the vertices s and t, and $\sigma_{st}(v)$ is the number of shortest paths between s and t that v lies on.

[13] Height threshold=0.2, max island size=7000.

[14] Strength subclass links=7, strength property links=0.2, strength definition links=3.

The basic problem of evaluating a partitioning is the fact, that in most cases we do not have an optimal partitioning to compare to. For these cases, we have to rely on alternative methods to determine the quality of the partitioning. A possibility that we will explore is empirical evaluation through user testing. Such an evaluation requires that the subjects have some knowledge about the domain modeled by the ontology. Therefore the ontology and the subjects have to be chosen carefully. The first option is to chose an ontology about a rather general topic (e.g. the transportation ontology). In this case any student is knowledgable enough to be chosen as a test subject. The other option is to chose a more specialized model and look for domain experts. Options here are the use of a computer science specific ontology (eg. the ACM classification) or a medical ontology. The advantage of the former is that test subjects are easier available while the time of medical experts is often rather limited.

A basic problem of empirical evaluation is the complexity of the task. Users will often not be able to oversee the complete ontology and to determine a good partitioning for themselves (in fact this is the reason why we need automatic partitioning). The most basic way of doing empirical evaluation is to directly use the notion of intra-pairs. As we have seen above, knowing all intra-pairs is sufficient for determining the quality measures defined above. This means that we can present pairs of concepts to subjects and ask them whether or not these concepts should be in the same part of the ontology. A problem of this approach is that the subject is not forced to be consistent. It might happen, that according to a subject A and B as well as A and C should be in the same part, but B and C should not. The second problem is the number of tests necessary to determine a partitioning. In the case of the ACM hierarchy, more that 1,5 Million pairs would have to be tested. In order to avoid these problems of consistency and scalability of empirical evaluation, we decided to perform an evaluation that is not based on concept pairs. The setting of our experiments is described in the following.

Setting

We used the ACM classification of computer science topics as a basis for performing an empirical evaluation of our partitioning method. The nature of the ACM hierarchy allows us to evaluate our method in terms of the number of key concepts identified when partitioning the model. The idea is that the root of each subtree distinguished by our partitioning algorithm should denote a unique subfield of computer science. When partitioning richer ontologies the hierarchy of one part would be a forest and centrality measures would be a better choice for denoting a subfield. However, for a partitioned hierarchy the subhierarchies are connected and the root is its superordinate concept. In order to determine such subfields that should be identified by the method, we analyzed the organization of computer science departments of Dutch universities with respect to the topics they used to identify subdivisions of their department. We then manually aligned these topics with the ACM topic hierarchy by translating the topic found into terms appearing in the ACM topic hierarchy. In cases where the topic matched more than one ACM terms (e.g. databases and information systems) both terms were counted. Terms that do not have a counterpart in the ACM hierarchy were ignored (e.g. 'mediamatics').

The test set consisted of 13 Dutch universities. Ten out of these had computer science departments. We extracted 85 terms from the corresponding web sites, mostly names of departments, groups or institutes. We were able to map 77 of these terms into 42 distinct terms from the ACM hierarchy. We distinguish three subsets of these 42 terms: terms that occur at least once, terms that occur at least twice and terms that occur at least three times. We can assume that terms that occur more than once to be important subfields of computer science that we would like to capture in a single module.

Results

We compared these extracted terms with the root concepts of subtrees of the ACM hierarchy generated using our partitioning method. We chose to use a setting where the maximal size of an island is set to 100 and the threshold for merging islands is 0.2. With these settings, the method generated 23 modules. We decided to ignore three of the root terms:

ACM CS Classification: This is the root of the hierarchy and not a proper term denoting a computer science topic

Mathematics of Computation: The subtopics of this will normally be found in mathematics rather than computer science departments and were therefore not covered by our test set.

Hardware: The subtopics of this module will normally be found in electrical engineering rather than computer science departments.

After this normalization, we compared the root terms of the generated modules given in Table 7.1 with the terms identified on the department web pages and used overlap to compute the quality of the partitioning in terms of precision and recall of our method.

From the web pages of Dutch computer science departments, we extracted the 42 ACM terms shown in Table 7.2. The most often occurring term was 'Algorithms' that described 5 groups, followed by 'Software' and 'Software Engineering'. Other frequently appearing topics were 'Robotics, 'Computer Systems', 'Computer Graphics', Information Systems', 'Expert Systems and Applications' (often referred to as 'Intelligent Systems'), Life Science applications, 'Systems Theory' and 'Theory of Computation'.

We can see that there is quite some overlap between the root nodes of the subtrees determined by our methods and the terms from the test set. The overlap is especially striking when we only consider the set of terms that occurred more than two times in the description of groups. Six out of these eleven terms where also determined by our method. The recall becomes worse when considering terms than only occurred twice or once. This was expected, however, because there are single research groups on more specific topics such as distributed databases that are not necessarily regarded as important subfields by a large majority of people. We included these terms with

Table 7.1. The method determined 20 terms to represent important subareas of computer science. (Apart from the three nodes Mathematics of Computing, ACM CS Classification and Hardware).

1. Numerical Analysis
2. Image Processing and Computer Vision
3. Management of Computing and Information Systems
4. Computing Milieux
5. Software Engineering
6. Computer Communication Networks
7. Data
8. Information Storage and Retrieval
9. Operating Systems
10. Database Management
11. Computer Systems Organization
12. Information Interfaces and Presentation
13. Software
14. (Mathematics of Computing)
15. Theory of Computation
16. (ACM CS Classification)
17. Information Systems
18. Computer Applications
19. Simulation and Modeling
20. Artificial Intelligence
21. Computer Graphics
22. Computing Methodologies
23. (Hardware)

less support in the test set to evaluate how many of the terms found by our method are used to describe the topics of groups. It turns out that 12 out of the 20 terms occur in the test set leading to a maximal precision of 60% for the largest test set. We used to F-Measure $((2 * (precision * recall))/(precision + recall))$ to determine the overall quality of the results. It turns out that we receive the best results on the set of terms that occur at least twice. A summary of the results is shown in Table 7.3.

The main observation is that there is a significant overlap between topics that occur in the name of computer science research groups and the root nodes of the subtrees determined by our method. We were able to reach a precision of up to 60 percent when considering all terms occurring on the web sites. When only considering terms that are used more than two times, our method reached a recall of almost 55 percent. This can be considered a very good result as the chance of picking the most frequently occurring terms from the ACM hierarchy is $\binom{11}{1300}$ (the binomial of 11 over 1300) and we do not have more information than the pure structure of the concept hierarchy.

This result supports our claim, that the structure of concept hierarchies contains important information about key concepts that in turn can be used to partition the hierarchy. Our hypothesis is, that this phenomenon is not random, but that people,

Table 7.2. ACM terms extracted from web sites of Dutch Computer Science Departments

occurrence	**ACM term**
> 2	Algorithms
	Software
	Software Engineering
	Robotics
	Computer Systems Organization
	Computer Graphics
	Information Systems
	Applications And Expert Systems
	Life And Medical Sciences
	Systems Theory
	Theory Of Computation
> 1	User Interfaces
	Programming Techniques
	Artificial Augmented And Virtual Realities
	Artificial Intelligence
	Image Processing And Computer Vision
	Input/Output And Data Communications
	Parallelism And Concurrency
	Probability And Statistics
> 0	Computer-Communication Networks
	Business
	Computing Methodologies
	Control Design
	Decision Support
	Distributed Artificial Intelligence
	Distributed Databases
	Formal Methods
	Games
	Information Search And Retrieval
	Information Theory
	Management Of Computing And Information Systems
	Microcomputers
	Natural Language Processing
	Neural Nets
	Numerical Analysis
	Physical Sciences And Engineering
	Real-Time And Embedded Systems
	Security
	Signal Processing
	Software Development
	System Architectures
	Systems Analysis And Design

Table 7.3. Summary of evaluation results

Test Set	Precision		Recall		F-Measure
> 2	30%	6 of 20	54.55%	6 of 11	38.71%
> 1	40%	8 of 20	42.11%	8 of 19	41.03%
> 0	60%	12 of 20	28.57%	12 of 42	38.71%

when creating classification hierarchies are more careful when determining the subclasses of important classes. The result is a high number of children that cause our method to split the hierarchy at this particular point.

7.4.3 Visualization and Reasoning via Module Extraction

Large ontologies often cause problems for reasoning and editing. Furthermore the time needed for a human to overlook an ontology dramatically increases with its size. If not the whole ontology but only parts of it are relevant for an application the straight forward approach to dealing with too large ontologies is to consider only a part of it.

[5] describes a scenario in which knowledge is selected from online available ontologies. This knowledge selection procedure was applied to the semantic web browser plugin Magpie [7]. Magpie requires to automatically select and combine online available ontologies for identifying and highlighting instances of concepts in associated colors. Figure 7.10 depicts the sequence of tasks that have to be performed a detailed desciption of the application can be found in Chap. 3.1.

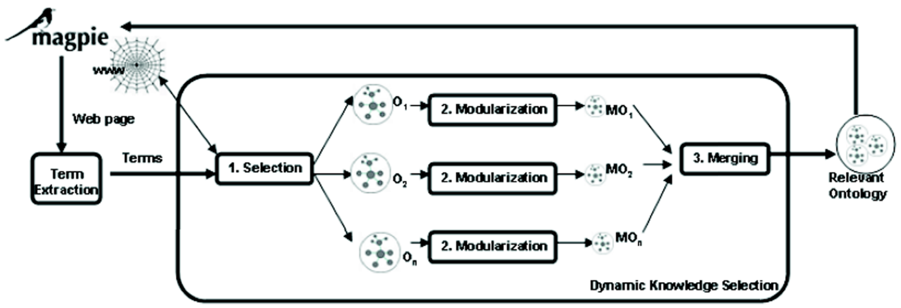


Fig. 7.10. The knowledge selection process and its use for semantic browsing with Magpie. Illustration from [6].

The partitioning method adressed in this chapter was applied for step 2, the extraction of relevant modules from the previously selected ontologies. A comparision of different extraction methods for this scenario is reported in [6], here we demonstrate how Pato is applied for the extraction process.

Usually knowledge extraction tools rely on a traversal approach and gather information from an ontology starting from a set of relevant terms. Nevertheless, in cases where knowledge is extracted repeatedly from the same ontology using a partitioning tool may be more efficient. The partitioning can be computed offline, repeated traversal of the whole ontology is replaced by the less complex selection of modules.

Setting

The scenario described above was simulated by manually extracting relevant keywords in news stories, using ontology selection tools[15]. In an example first described in [11] the keywords *Student*, *Researcher*, and *University* where used to select ontologies. Three ontologies covering these terms where obtained:

ISWC: http://annotation.semanticweb.org/iswc/iswc.owl
KA: http://protege.stanford.edu/plugins/owl/owl-library/ka.owl
PORTAL: http://www.aktors.org/ontology/portal

The appropriateness of Patos partitionings for this application is evaluated by two new criteria. First, the size of the obtained modules should be small with respect to the original ontology. Second partitioning with all relevant terms in one module are prefered i.e. the number of modules containing relevant terms should be small.

$$\textit{relativeSize=-40}$$
$$\textit{numberOfRelevantModules}^3\textit{=-1}$$

The former criterion is the relative size of the resulting module compared to the size of the ontology, the latter criterion was emphazised by an exponent. In this application the relative importance of the two criteria can not be modelled by linear weights only because its gradient is not zero. By setting the exponent to 3 we make sure the rating of a partitioning is more affected by the decreasing the number of relevant modules from e.g. 3 to 2 than by the decreasement from 2 to 1.

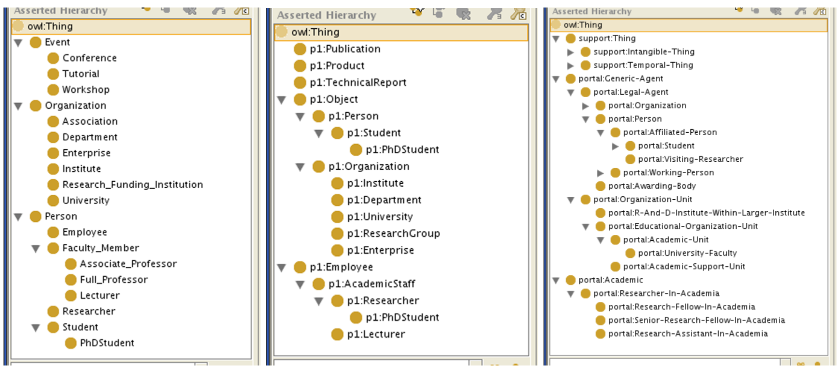
Results

Table 7.4 shows that for ISWC and KA the merging optimization was not necessary (the height ranges between 0 and 1, so merging modules with heigth 1.1 and larger means not merging at all). Due to the application requirement of small module size leftover nodes and small modules are not merged into larger modules. The low height

Table 7.4. Configuration and evaluation for the knowledge extration setting

		ISWC	KA	PORTAL
configuration	max island size	40	10	15
	height threshold	1.1	1.1	0.2
	strength definition links	1	0	0
evaluation	relevant modules	1	2	1
	relative Size	0.54	0.14	0.27

[15] In particular Swoogle (http://swoogle.umbc.ed).



```
Asserted Hierarchy
owl:Thing
▼ Event
    Conference
    Tutorial
    Workshop
▼ Organization
    Association
    Department
    Enterprise
    Institute
    Research_Funding_Institution
    University
▼ Person
    Employee
  ▼ Faculty_Member
      Associate_Professor
      Full_Professor
      Lecturer
    Researcher
  ▼ Student
      PhDStudent
```

```
Asserted Hierarchy
owl:Thing
  p1:Publication
  p1:Product
  p1:TechnicalReport
▼ p1:Object
  ▼ p1:Person
    ▼ p1:Student
        p1:PhDStudent
  ▼ p1:Organization
      p1:Institute
      p1:Department
      p1:University
      p1:ResearchGroup
      p1:Enterprise
▼ p1:Employee
  ▼ p1:AcademicStaff
    ▼ p1:Researcher
        p1:PhDStudent
      p1:Lecturer
```

```
Asserted Hierarchy
owl:Thing
▼ support:Thing
  ▶ support:Intangible-Thing
  ▶ support:Temporal-Thing
▼ portal:Generic-Agent
  ▼ portal:Legal-Agent
    ▶ portal:Organization
    ▼ portal:Person
      ▼ portal:Affiliated-Person
        ▶ portal:Student
          portal:Visiting-Researcher
      ▶ portal:Working-Person
      portal:Awarding-Body
  ▼ portal:Organization-Unit
      portal:R-And-D-Institute-Within-Larger-Institute
    ▼ portal:Educational-Organization-Unit
      ▼ portal:Academic-Unit
          portal:University-Faculty
        portal:Academic-Support-Unit
▼ portal:Academic
  ▼ portal:Researcher-In-Academia
      portal:Research-Fellow-In-Academia
      portal:Senior-Research-Fellow-In-Academia
      portal:Research-Assistant-In-Academia
```

Fig. 7.11. Relevant modules of ISWC, KA and PORTAL ontologies for the terms researcher, student, university

threshold for PORTAL is caused by the second criterion. For forcing all terms into one module more merging was necessary.

7.5 Conclusion

In this chapter we have described a method for structure-based ontology partitioning that is practically applicable to very large ontologies. The main idea of the algorithm is translating the structure of the ontology to a weighted graph. This graph is split up such that the resulting modules are stronger internally connected than externally connected. Finally three optimization steps are performed to improve the partitioning. Experiments on different ontologies have shown that a modularization based only on structural properties of the ontology already results in modules that intuitively make sense. Helpful visualization of large ontologies and extraction of key topics provided evidence for the appropriateness of the proposed approach.

Because modularizing an ontology essentially is a modeling activity, there is no “golden standard” to compare our results with. Actually, the notion of a “good” partitioning depends to a large extent on the application that uses the partitioned ontology. For enabling adaption to different application requirements we designed a parameterized partitioning algorithm. The parameters that determine the resulting partitioning are set automatically depending on given requirements. Encoding parameter selection as an optimization problem with respect to relevant quality criteria has been crucial for implementing automatic parameter selection. This disburdens the user from thinking about technical details of the algorithm and draws his attention to the requirements of the application at hand.

In future work we are planning to use our partitioning tool for distributed reasoning. The performance of distributed reasoning algorithms depends on additional properties like the size of the shared language. These requirements will be evaluated and implemented in the further development process of Pato. Thus it will be possible to configure the partitioning process to result in a reasonable trade-off between reasoning related requirements and maintainability requirements.

References

1. Anquetil, N., Fourrier, C., Lethbridge, T.C.: Experiments with hierarchical clustering algorithms as software remodularization methods. In: Proceedings of the Working Conference on Reverse Engineering (WCRE 1999), Atlanta, USA, October 1999, pp. 304–313 (1999)
2. Batagelj, V.: Analysis of large networks - islands. In: Dagstuhl seminar 03361: Algorithmic Aspects of Large and Complex Networks (August/September 2003)
3. Broekstra, J., Kampman, A., van Harmelen, F.: Sesame: A generic architecture for storing and querying RDF and RDF schema. In: Horrocks, I., Hendler, J. (eds.) ISWC 2002. LNCS, vol. 2342, pp. 54–68. Springer, Heidelberg (2002)
4. Burt, R.S.: Structural Holes. The Social Structure of Competition. Harvard University Press (1992)
5. d'Aquin, M., Sabou, M., Motta, E.: Modularization: a key for the dynamic selection of relevant knowledge components. In: Workshop on Modular Ontologies (WoMO) (2006)
6. d'Aquin, M., Schlicht, A., Stuckenschmidt, H., Sabou, M.: Ontology Modularization for Knowledge Selection: Experiments and Evaluations. In: Wagner, R., Revell, N., Pernul, G. (eds.) DEXA 2007. LNCS, vol. 4653, pp. 874–883. Springer, Heidelberg (2007)
7. Dzbor, M., Domingue, J., Motta, E.: Magpie - towards a semantic web browser. In: Fensel, D., Sycara, K.P., Mylopoulos, J. (eds.) ISWC 2003. LNCS, vol. 2870, pp. 690–705. Springer, Heidelberg (2003)
8. Freeman, L.C.: A set of measures of centrality based on betweenness. Sociometry 40(1), 35–41 (1977)
9. Mitchell, B.S., Mancoridis, S.: Comparing the decompositions produced by software clustering algorithms using similarity measurements. In: Proceedings of the 17th IEEE International Conference on Software Maintenance (ICSM 2001), Florence, Italy, November 2001, pp. 744–753 (2001)
10. Ogren, P.V., Cohen, K.B., Acquaah-Mensah, G., Eberlein, J., Hunter, L.: The compositional structure of gene ontology terms. In: Pacific Symposium on Biocomputing (2004)
11. Sabou, M., Lopez, V., Motta, E.: Ontology selection on the real semantic web: How to cover the queens birthday dinner? In: Staab, S., Svátek, V. (eds.) EKAW 2006. LNCS, vol. 4248, pp. 96–111. Springer, Heidelberg (2006)
12. Schlicht, A., Stuckenschmidt, H.: Towards Structural Criteria for Ontology Modularization. In: Workshop on Modular Ontologies ISWC (2006)
13. Stuckenschmidt, H., Klein, M.: Structure-based partitioning of large concept hierarchies. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 289–303. Springer, Heidelberg (2004)

8

Web Ontology Segmentation: Extraction, Transformation, Evaluation

Julian Seidenberg

Bio-Health Informatics Group,
University of Manchester, UK
j@deltaflow.com

Summary. In this chapter we present an algorithm for extracting relevant segments out of large description logic ontologies for the purposes of increasing tractability for both humans and computers. We offer several variations on this algorithm for different purposes. The segments are not mere fragments, but stand alone as ontologies in their own right. This technique takes advantage of the detailed semantics captured within an OWL ontology to produce highly relevant segments. However, extracted segments make no guarantee for preserving the semantics of the complete ontology.

8.1 Introduction

The research presented in this chapter uses the large and complex GALEN ontology of medical terms and surgical procedures (produced during the 1990s by the University of Manchester in the OpenGALEN project [34] [30]) as a test platform for such an ontology segmentation algorithm. Since the complete GALEN ontology was only available in its own proprietary format, it was converted into an OWL representation for the purposes of this research. Only small, incomplete versions of GALEN in OWL have previously been available[1].

A basic algorithm for segmentation by traversal is presented and evaluated. This algorithm is modified to limit the depth of the recursive evaluation. This so-called *boundary-depth limited* algorithm minimizes the size of the resulting segments.

The algorithm is further modified by filtering certain property relations. This *property filtered* algorithm increases the tractability of the ontology for classification by description logic reasoning systems.

The algorithm is also adapted to transform the extracts during segmentation. Different techniques for creating effects of *transitive propagation* [29] are applied while the extracted segments are created. The classification performance of these

[1] The complete GALEN in OWL along with a web application that can generate custom GALEN segments is available online at `http://www.co-ode.org/galen`

different transformed segments is evaluated. This technique shows how transforming an ontology while segmenting can be used to increase ontology performance [39].

Finally, an overview of various alternative approaches is given, grouping the segmentation techniques into segmentation by query, network partitioning and segmentation by traversal [40].

8.1.1 The Problem of Large Ontologies

Ontologies can add tremendous value to web technologies. As Jim Hendler has pointed out on numerous occasions "a little semantics goes a long way" [12]. The knowledge captured in ontologies can be used, among other things, to annotate data, distinguish between homonyms and polysemy, generalize or specialise concepts, drive intelligent user interfaces and even infer entirely new (implicit) information.

The ultimate vision for a semantic web is to create a world wide web that computers can understand and navigate. Making this vision a reality will either require an extremely large ontology that describes every term of interest on the Internet, or, more realistically, numerous domain-specific ontologies, which, when aligned with one another, form a web of semantic inter-ontology relations. Either way, the result is a very large knowledge corpus.

Examples of such enormous ontologies are already starting to appear. For example, the biomedical domain has numerous very large ontologies such as SNOMED-CT [43], GALEN [30], FMA [36] and NCI-Thesaurus [9]. However, these ontologies have grown too large to effectively used and maintained, often requiring large teams of highly trained experts [44].

If a truly massive semantic web is going to be of use to anyone, users and applications will have to find a way to limit their scope. The knowledge web, as a whole, will be too big and mostly irrelevant for any single task.

8.1.2 Solutions to the Scaling Problem

Google solves the problem of scaling web search by creating partially sorted barrels of keyword indexes. Searches are distributed over a very large cluster of computers [5]. A similarly sophisticated distributed system may be feasible for use with the ontologies of the semantic web. However, ontologies, such as those represented in the Web Ontology Language (OWL) [22], are significantly more complex data structures than mere web pages. OWL builds several levels of complexity on top of the XML of conventional web data [4] [15]. It is likely that large and complex ontologies will require a novel solution.

We suggest such a solution: instead of attempting to capture the entire semantic web in a gigantic index, each web application extracts and uses a custom ontology segment specific to its particular needs. Segments are big enough to be useful, but not so big that scaling becomes a problem.

The ontology segmantation techniques shown in this chapter exploit the semantic connections between ontology terms and thereby enable web-application developers to quickly (or even automatically) create the custom ontologies they require. This is a first step towards a working application layer on top of a large-scale semantic web.

8.1.3 Other Applications for Segmentation

Custom ontology segments, as described above, show potential for a wide variety of use cases. For example:

- Query efficiently could be substantially improved by querying segments instead of querying the complete ontology network.
- Segments could be used as examples of and discussion points for specific modeling patterns.
- Segments could be captured at specific time points as backup or provenance data.
- Similar segments from different ontologies in the same domain could be used for comparison and evaluation purposes.
- Segmentation could be used to specify, outline and annotate specific ontology sub-sections.
- Segments from general purpose ontologies could be transformed on-the-fly during the extraction process to produce optimal ontologies for a specific applications.

8.1.4 Scope

The algorithm presented in this chapter is optimized to work with knowledge bases similar to the GALEN ontology. That is, a large ontology with over 1000 classes and dense connectivity, with at least, on average, one restriction asserted per concept.

Another pre-requisite for our segmentation methodology is that the ontology be normalised [31]. Normalisation greatly simplifies ontology maintenance.

Primitive classes in a normalised ontology have no more than one primitive superclass: multiple parents are modeled implicitly and left to be explicitly inferred later. This is done because a mono-hierarchy is much easier to maintain than a poly-hierarchy. When using normalisation the reasoner can do the hard work of creating the poly-hierarchy.

Defined classes have exactly one named class in their definition. This named class can be viewed as a primitive class' single superclass and so the segmentation algorithms presented herein can work equally well on both primitive and defined concepts.

GALEN in OWL uses the $\mathcal{SHIF}$ subset (without negation or disjunction) of the full $\mathcal{SHOIN}(\mathcal{D})$ expressivity of OWL-DL, so the segmentation is currently constrained to that. The methodology presented here is not meant to offer a complete solution with rigorous logical proofs. Instead, we present empirical evidence as to the effectiveness of our approach.

Most ontologies' properties are structured as flat lists. GALEN however employs a rich property hierarchy with over 500 distinct properties. This is especially useful for producing extracts constrained to specific user and/or application requirements. Ontologies with simple property structures, such as, for example, the Gene Ontology [42], will not be able to take advantage of this aspect of the segmentation algorithm presented herein.

We do not consider instances in our segmentation algorithm.

8.1.5 Aim: Useful Classification and Small Size

Description logic reasoners such as FaCT++ [47], RACER [11], or Pellet [27] can be used to infer new information that is implicit in an ontology [19]. This process is very important, especially for an ontology like GALEN, which was built with normalisation principles in mind. It is therefore critical that GALEN in OWL can be classified.

However, none of the above mentioned description logic reasoners based on the tableaux algorithm are currently able to classify the complete GALEN ontology. GALEN is too large and complex for these reasoning systems. A primary aim of this research was therefore to produce classifiable segments. The ideal segment is as small and focused as possible, while still containing enough information to enable the reasoner to infer relevant new subsumption relationships. However, as will be seen later, certain segmentation techniques can result in incomplete classification results.

8.2 Links in Description Logic

8.2.1 Restrictions as Links

OWL ontologies usually contain large hierarchies of concepts. They also feature the ability to add *restrictions* to such concepts. The most common types of *restrictions* restrict the *individuals* that a certain class describes. These *restrictions* are *quantified* by, for example, the *existential* ($\exists$) or *universal* ($\forall$) quantifiers. *Quantified restrictions* also include a *property* and *filler concept* to specify how the members of a class are restricted.

Restrictions, from one point-of-view, are anonymous classes and can be added as superclasses of another (named) class. For example: the class *MalePerson* might have the restriction in Figure 8.1 asserted as its superclass. This means that all individuals that the *MalePerson* class defines must have one or more relations using the *hasGender* property to individuals in the class *MaleGender*. Figure 8.1 illustrates this relationship.

However, seen from another point-of-view, restrictions represent cross-links between different classes as shown in Figure 8.2, so that an ontology can be seen as a large hierarchy of classes linked by restrictions.

In reality, of course, the anonymous qualified restriction superclasses actually restrict individuals' relations to other individuals, but it is useful to think of them simply as links. We will, from here on, assume this model of ontological topology.

8.2.2 Reciprocal Links

Besides normal forward links, as described above, backward links, or usages, also known as *reciprocal links*, are also important in understanding the structure of an ontology.

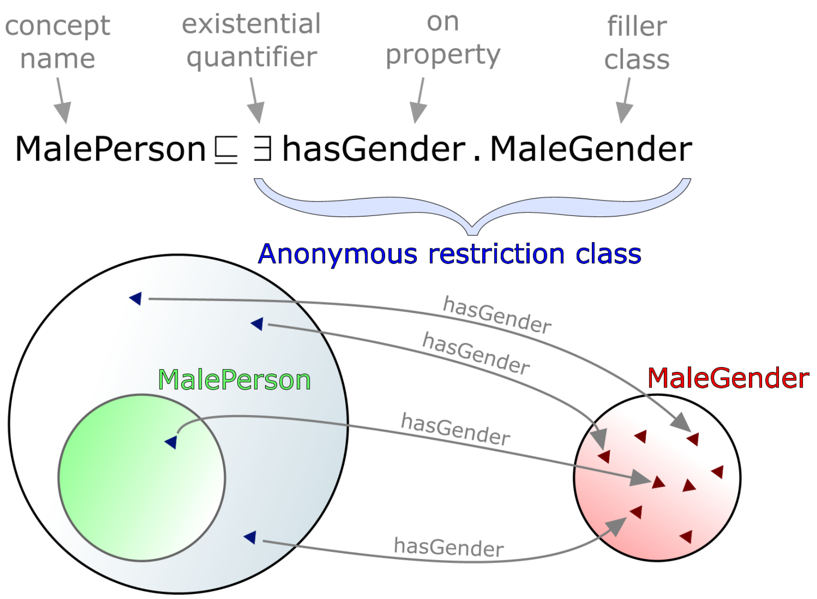


Fig. 8.1. Superclass restriction and the corresponding links between individuals

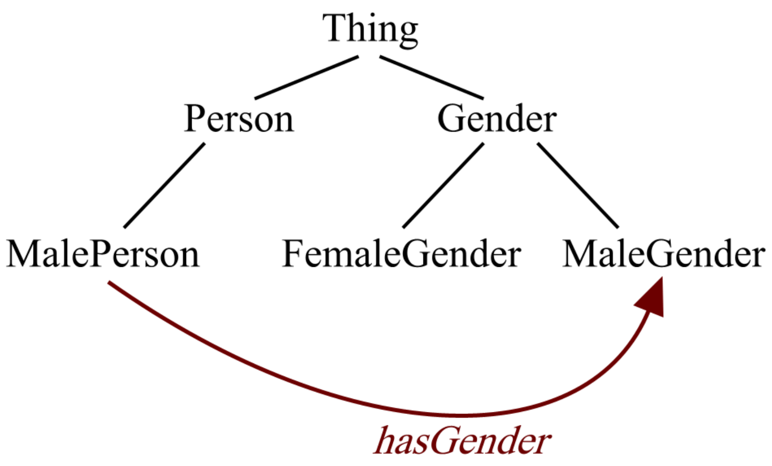


Fig. 8.2. Interpreting quantified restrictions as links between classes

Even though "isPartOf" and "hasPart"are inverses of each other, the *reciprocal* statements in Figure 8.3 are not equivalent; in fact neither implies the other.

GALEN is unusual in that it commonly represents anatomy using *reciprocal* pairs of restrictions. This representation is inherently cyclical and connects every piece of anatomy with every related piece in both directions. Tableaux classifiers

$Finger \sqsubseteq \exists$ *isPartOf* $. Hand$
(*all fingers are part of some hand*)

$Hand \sqsubseteq \exists$ *hasPart* $. Finger$
(*all hands have some finger as part*)

Fig. 8.3. Example of a reciprocal link

intrinsically scale exponentially when faced with such constructs. None of the current tableaux-based reasoners listed in Section 8.1.5 can classify even a small extract of the GALEN ontology containing both types of *reciprocal links* present in the original. (Note: the original classifier used in GALEN used different principles and did not suffer from this particular limitation [16].)

The algorithm presented herein therefore takes the approach of taking all reciprocals into account, but producing actual segments with only one-way links, using, for example, only "isPartOf" relations. Some of the new relations may be virtual: i.e. have only been found by first adding the reciprocals.

$Finger \sqsubseteq \exists$ *hasPart* $. Hand$
(*all fingers have some hand as part*)

$Hand \sqsubseteq \exists$ *hasPart* $. Finger$
(*all hands have some finger as part*)

Fig. 8.4. Example of a symmetric link

It is important to note that these reciprocal links differ from symmetric links. That statement in Figure 8.4 is a symmetric link, which has a very different meaning to the example of a reciprocal link given above. Symmetric links do not adversely affect classification.

8.3 Basic Segmentation Algorithm

The basic segmentation algorithm starts with one or more classes of the user's choice and creates an extract based around those and related concepts. These related classes are identified by following the ontology link structure.

8.3.1 Upwards Traversal of the Hierarchy

Assuming, for example, that a segment of the *Heart* class is to be produced. The obvious first class to include is the *Heart* class, the *Heart's* superclass (InternalOrgan), then that class' superclass and so on, all the way up the hierarchy, until the *top* ($\top$) concept is reached. Since this hierarchy is often quite deep (13 superclasses in this case) one might consider collapsing the tree by merging several superclasses. However, this destroys some of the semantic accuracy of the ontology. It may be sensible when constructing an ontology *view* or *perspective*, but is not useful for any extract that is to be used in an application (such as a classifier), since each superclass might contain critical information.

8.3.2 Downwards Traversal of the Hierarchy

The algorithm also goes down the class hierarchy from the *Heart* class, including its subclasses (in this case: UniventricularHeart). This is especially relevant when segmenting an ontology that has already been classified where newly inferred subclasses of a particular class are likely to be of interest.

The property hierarchy is however never traversed downwards. Properties are not of interest unless they are used in the class hierarchy. So, if they are used, they, their superproperties and no other properties, are included.

8.3.3 Sibling Classes in the Hierarchy

Sibling classes are not included in the extract. The *Heart* class' siblings include concepts like the *Lung*, *Liver* and *Kidney*. It is reasonable to assume that these are not relevant enough to be included by default. The user can always explicitly select them for inclusion, if they are of interest.

8.3.4 Upwards and Downwards and Upwards from Links

Having selected the classes up & down the hierarchy from the target class, their restrictions, intersection, union and equivalent classes now need to be considered: intersection and union classes can be broken apart into other types of classes and processed accordingly. Equivalent classes (defined classes which have another class or restriction as both their subclass and their superclass) can be included like any other superclass or restriction, respectively. Restrictions generally have both a *type* (property) and a *filler* (class), both of which need to be included in the segment.

Additionally, the superproperties and superclasses of these newly included properties and classes also need to be recursively included, otherwise these concepts would not have a place in the subsumption hierarchy and would all end up as subclasses of *OWL:Thing*. That is, without being attached to the hierarchy, concepts are assumed to simply be subsumed by the *top* concept ($\top$), leading to a very messy, confusing and often intuitively incorrect view of the unclassified ontology.

Figure 8.5 gives an illustration of this segmentation algorithm. Starting at the target of the extract, the algorithm traverses the hierarchy upwards all the way to the root class. It also traverses it downwards all the way to the leaf classes. Additionally, any links across the hierarchy from any of the previously traversed classes are followed. The hierarchy is traversed upwards (but not downwards) from any of these classes that the cross-links point to. Links pointing at other classes from these newly traversed classes are also included. This continues until there are no more links left to follow.

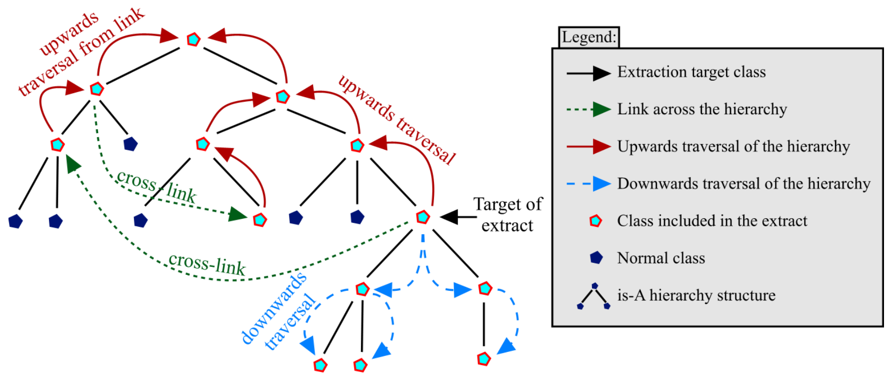


Fig. 8.5. Traversal Up & Down and Up from links

8.3.5 But Not Downwards from Upwards Links

Finally, one might also consider including the subclasses of those classes included via links. However, doing so would result in including the entire ontology. This is something one definitely wants to avoid when creating an extract.

8.4 Constraining Segment Size

The segmentation algorithm outlined above produces an extract of all concepts related to the target concept. However, with densely interconnected ontologies, such as for example GALEN, this new ontology is usually only up to one fifth the size of the original. A means of further constraining segments is needed.

8.4.1 Property Filtering

If the aim is to produce a segment for use by a human, or specialized application, then filtering on certain properties is a useful approach.

For example, if a user is not interested in the diseases modeled in GALEN, he or she can specify to exclude all *locative properties*. These are properties that specifically link diseases to the locations in the body where they might occur: e.g. “IschaemicCoronaryHeartDisease *hasLocation* Heart”.

The upper-level meta-properties which it may be logical to include and/or exclude will be different for each ontology to be segmented. These meta-properties are, in this case, actual properties, since GALEN groups related properties together under super-properties. The following meta-properties and their inverses were selected for course grain property filtering:

- **modifierAttribute:** properties which can be used to modify a given class such as “colour” or “status”. These are sometimes also known as “value partitions” [7]. They are almost always safe to include in an extract, since the class values they link to do not themselves link onwards to other classes and therefore will not significantly increase a segment's size.

- **constructiveAttribute:** the super-property of all the following properties.
  - **locativeAttribute:** properties that link diseases to anatomical locations that they are in some way related to.
  - **structuralAttribute:** properties linking anatomical body structures together by physical composition.
  - **partitiveAttribute:** properties that link classes based on processes, divisions and other partitive relations
  - **functionalAttribute:** properties that link classes by action or function.

Note: The various properties could be broken down much more elaborately. However, the point is that organizing properties under any sensible upper-level property structure will enable some degree of useful property filtering. A more detailed analysis of the GALEN property hierarchy may be found in [35].

Removing trivially equivalent definitions

Properties are filtered by removing all restriction in which they occur. However, upon removing such restrictions from defined class, it frequently occurs that a definition becomes indistinguishable and therefore equivalent to another similar definition. The resultant long chains of equivalent classes, while not wrong in the context of the filtered ontology segment, are not what was intended in the original base ontology. They are also difficult to view in ontology editors such as Protégé OWL [18]. Trivially equivalent definitions are therefore transformed into primitive classes by the segmentation algorithm. These still occupy the correct place in the hierarchy and are easy for editors to display.

$$\mathit{SkinOfFrontalScalp} \equiv \begin{pmatrix} \mathit{SkinOfScalp} \sqcap \\ \exists\, \mathit{hasSpecificProximity} \,.\, \mathit{FrontalBone} \end{pmatrix}$$

$$\mathit{SkinOfFrontalScalp} \equiv \mathit{SkinOfScalp}$$

$$\mathit{SkinOfFrontalScalp} \sqsubseteq \mathit{SkinOfScalp}$$

Fig. 8.6. Property filtering with trivial definition removal

As shown in the progression in Figure 8.6, if the filtering process removes the restriction on a class and this results in a trivial equivalence, then the definition is converted into a primitive class.

8.4.2 Depth Limiting Using Boundary Classes

Depth limiting is a useful approach for accurately adjusting the size of a segment so that it can, for example, be classified successfully by automated reasoners.

A chain of links is followed to create a list of classes to include in an extract. In doing so, each classes' restrictions' filler classes should be included to produce an extract that retains the structure of the original (see Sections 8.2.1 and 8.3.4). However, if, upon reaching a certain recursion depth, calculated from the extract's target concept, all the links on a class are removed, this class becomes a *boundary class*.

$Heart \sqsubseteq \exists\, hasStructuralComponent\,.\,Pericardium$
$Pericardium \sqsubseteq$ *SerousMembrane*
$Pericardium \sqsubseteq$
$\quad\exists$ *isStructuralComponentOf*$\,.\,CardiovascularSystem$

Fig. 8.7. Example of a boundary class

For example, one might remove the axiom in Figure 8.7 stating that the *Pericardium* (the membrane that surrounds the heart) is a component of the *CardiovasuclarSystem* (line three of the Figure), since one may not be interested in including the *CardiovascularSystem* and everything related to it in a segment of the *Heart*. This creates a *boundary class* that is still defined in the hierarchy (under *SerousMembrane*) and therefore still makes sense within the ontology, but has an incomplete definition.

The named superclasses of a boundary class (line two of Figure 8.7) must be included in the extract in order to place classes in their proper position in the hierarchy. These classes would otherwise all be subsumed under the top concept ($\top$). These superclasses are however also boundary classes, unless they are linked to by way of shorter recursion path along another concept, as shown in Figure 8.8.

The main hierarchy of "is-A" superclass relationships between classes should not be counted when calculating the traversal depth, since they need to be included in any case and do not substantially increase the complexity of the segment. Subclass relations can be ignored completely, since they are not included in the extract in the first place (see Section 8.3.5). Figure 8.8 illustrates the entire boundary extraction procedure.

This methodology effectively limits the size of the ontology, since the presence of a boundary class will cause a link traversal algorithm to terminate. Otherwise, in the case of a densely interlinked ontology such as GALEN, practically the entire ontology could be "linked-in".

Noy and Musen's ontology extraction research [24], also uses the *boundary class* term, but defines it as any class which is in the range of any property that is used in each restriction on each of the classes which are targeted by the extract. The resulting list of *boundary classes* is to function as a reminder to the user of other classes they might want to include in the view they are constructing. This approach relies on property ranges being specified in the ontology, which is often not the case and on a graphical user interface to "prompt" [23] the user. The approach presented here takes a more automated approach, aiming to produce a heuristic algorithm that creates a useful segment without much user intervention.

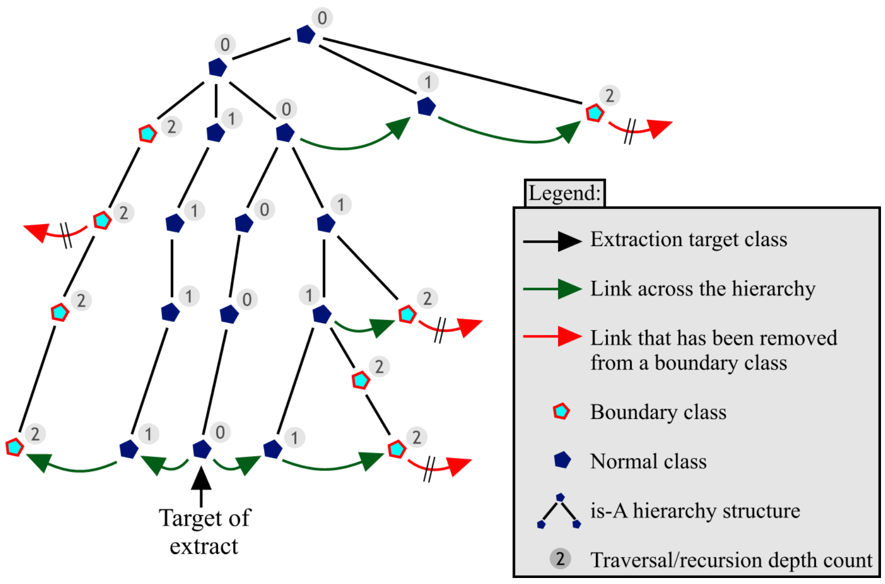

Target of extract
Legend:
Extraction target class
Link across the hierarchy
Link that has been removed from a boundary class
Boundary class
Normal class
is-A hierarchy structure
Traversal/recursion depth count

Fig. 8.8. Boundary extract with depth limited to 'two'

8.5 Transitive Propagation Transformation

This section describes transitive propagation and different ways it can be approximated in description logic. These different techniques for representing transitive propagation are then applied as transformations during segmentation process.

8.5.1 Transitivity

A transitive relation is a relation between three elements if it holds between the first and second and it also holds between the second and third it must necessarily hold between the first and third [45].

Transitivity is one of the three intrinsic properties of part/whole relations. Winston calls this "a single sense of part" [49]: if the door is part of the car and the door-handle is part of the door, then the door-handle is also part of the car. If (A *isPartOf* B) and (B *isPartOf* C) then (A *isPartOf* C).

8.5.2 Transitive Propagation

However, the above does not necessarily hold true universally. Odell [26] points out that there are many different kinds of composition. When we say "part of" we often mean very different things. For example, "Iron isPartOf Car" implies a material-object relation, i.e. the car object is made of the iron material, while "Car isPartOf Traffic" implies a member-bunch relation, i.e. the car is a member of the collection of things which make up the Traffic concept.

While each specific type of part/whole relation is transitive along relations with the same semantics, as illustrated in Figure 8.9, this does not necessarily hold true across different types of relations, as shown in Figure 8.10.

$$(Piston\ isPartOf\ Engine) \sqcap (Engine\ isPartOf\ Car) \rightarrow (Piston\ isPartOf\ Car)$$

Fig. 8.9. Transitive relation because of similar semantics

$$(Piston\ isPartOf\ Car) \sqcap (Car\ isPartOf\ Traffic) \nrightarrow (Piston\ isPartOf\ Traffic)$$

Fig. 8.10. Non-transitive relation because of different semantics

However, as will be explained in the next section, in some cases, *transitive propagation* (sometimes also called a *role path*, or *propagates-via*) along relations with different semantics is desireable.

(Note: up to this point we referred to all relations as "partOf" in order to illustrate transitivity. However, for the purpose of more clearly distinguishing between relations with different semantics, we will proceed to name them more descriptively.)

8.5.3 Styles of Transitive Propagation

A specific illustrative example will be used throughout this chapter to show the different styles of modelling transitive propagation in OWL. Assuming the knowledge base in Figure 8.11 is given:

$$Foot \sqsubseteq \exists\ isPartOf\,.\,Leg$$
$$Toe \sqsubseteq \exists\ isPartOf\,.\,Foot$$
$$Burn \sqsubseteq \exists\ isLocatedIn\,.\,Toe$$
$$LegInjury \equiv \exists\ isLocatedIn\,.\,Leg$$

Fig. 8.11. Initial example ontology

The defined class in the last line of Figure 8.11 serves as a query. It should subsume all possible injures to the *Leg* when the knowledge base is classified. That is, once all implicit relationships in the ontology are made explicit, and assuming transitive propagation is properly modeled, then *Burn* should be subsumed under *LegInjury*.

8.5.4 Property Subsumption

One way of simulating transitive propagation is to use the property hierarchy to assert one property as a subproperty of another. For example, if "*isLocatedIn* propagatesVia *isPartOf*", then "*isPartOf* subsumes *isLocatedIn*", where both *isLocatedIn* and *isPartOf* are transitive properties. This is shown more formally in Figure 8.12.

$$r \circ s \dot{\sqsubseteq} r \Rightarrow s \sqsubseteq r \quad (\circ \textit{ indicates transitive propagation})$$
$$s \in R_+ \qquad (R_+ \textit{ is the set of transitive property names})$$

Fig. 8.12. Simulating transitive propagation by using the property hierarchy

This method is easy to understand, simple to implement and, as will be shown later in section 8.7, provides good performance. However, it also has numerous disadvantages.

As pointed out by Rector in [31], a tangled ontology is very difficult to maintain. Tangled ontologies have subsumption hierarchies with more than one superclass per class. The maintenance difficulty is equally applicable to a hierarchy of properties. Using property subsumption to simulate transitive propagation in ontologies with large numbers of properties can therefore quickly lead to an unmaintainable knowledge base. In such cases, the information about mutual propagtion among properties is best kept externally and applied to ontology using a script. JOT [6], for example, is well suited for this purpose.

Another disadvantage of this method is that its logical meaning is inaccurate. Unexpected logical inferences are therefore sometimes possible. For example, given the knowledge bases in Figure 8.11 and the property subsumption method, as outlined above ($\textit{isLocatedIn} \circ \textit{isPartOf} \dot{\sqsubseteq} \textit{isLocatedIn} \Rightarrow \textit{isPartOf} \sqsubseteq \textit{isLocatedIn}$), the query for *LegInjury* would result in both *Burn* and *Toe*. That is, both concepts would be inferred as subclasses of *LegInjury*, since any *isPartOf* relation is also an *isLocatedIn* relation.

Inferring *Toe* as a subclass of *LegInjury* is obviously not the intended meaning, but may be acceptable in some cases. For example, further restricting the query by adding more information, as shown in Figure 8.13, yields the expected result. The property subsumption technique for transitive propagation certainly requires careful analysis and should never be applied blindly.

$$Burn \sqsubseteq Injury$$
$$Burn \sqsubseteq \exists\, \textit{isLocatedIn} \,.\, Toe$$
$$LegInjury \equiv \begin{pmatrix} Injury \sqcap \\ \exists\, \textit{isLocatedIn} \,.\, Leg \end{pmatrix}$$

Fig. 8.13. Correctly behaving LegInjury query using property subsumption

8.5.5 Classic SEP Triples

Schulz and Hahn introduce the idea of SEP triples [37]. Their idea allows transitivity to be modeled explicitly. That is, SEP triples enable transitive relations to be expressed in formalisms that do not include transitivity by explicitly distinguishing between the whole of a concept, parts of a concept and the disjunction of the whole of a concept and its parts. Details of these triples may be found in [37].

An implementation of classic SEP triples requires extensive modification of the ontology class hierarchy. Three separate classes need to be introduced for every actual concept in the knowledge base. This results in a complex ontology structure that is difficult to maintain. Furthermore, we do not know of any algorithm for creating an ontology with SEP triples from a base ontology, given a list of transitively propagating properties. The performance of classic SEP triples was therefore not evaluated as part of this research. However, we presume their performance is inferior to that of the adapted SEP triples methodology (see below), since they do not take full advantage of OWL.

8.5.6 Adapted SEP Triples

Rector suggests an adapted SEP triples formalism [29] for use in description logics with transitive properties such as OWL ($\mathcal{SHOIN}(\mathcal{D})$) [15].

Similar to classic SEP triples, this methodology explicitly models transitive propagation in the knowledge base. However, unlike Schulz and Hahn's original idea, adapted SEP triples take advantage of OWL's ability to represent transitive properties. This removes the need to model transitivity explicitly in the knowledge base and therefore allows a much cleaner SEP triple-like representation to be created.

Example

$$\begin{array}{ll}
\mathit{isLocatedIn} \in R & (R \textit{ is the set of all property names}) \\
\mathit{isPartOf} \in R_+ & (R_+ \textit{ is the set of transitive properties}) \\
R_+ \subseteq R & \\
\mathit{LegInjury} \equiv \exists\, \mathit{isLocatedIn}\,.\left(\begin{array}{c} \mathit{Leg} \sqcup \\ \exists\, \mathit{isPartOf}\,.\,\mathit{Leg} \end{array}\right) &
\end{array}$$

Fig. 8.14. Adapted SEP triples query

Reusing the example knowledge base from Figure 8.11 above and classifying it together with the the query in Figure 8.14, results in the the expected inference: *Burn* is found to be a subclass of *LegInjury*.

Figure 8.15 shows how this adapted SEP triples mechanism works:

Toe, *Foot* and *Leg* are all concepts that are transitively part of each other, as indicated by the solid, upwards arcing arrows. There is also the *Burn* concept located in the *Toe*. Additionally, the model contains *LegInjury*, which is the defined class from Figure 8.14 that captures all things located in the *Leg*, or any of its parts.

Since *isPartOf* is a transitive property, the *Toe* concept is also part of the *Leg* concept. Therefore, anything located in the *Toe* (such as the *Burn*) matches the second part of the definition of *LegInjury*. That is, it is located in something which is part of the *Leg*. This results in *Burn* being inferred as a subclass of *LegInjury* when the knowledge base is classified by a description logic reasoner, as indicated by the dotted, straight, upwards pointing arrow.

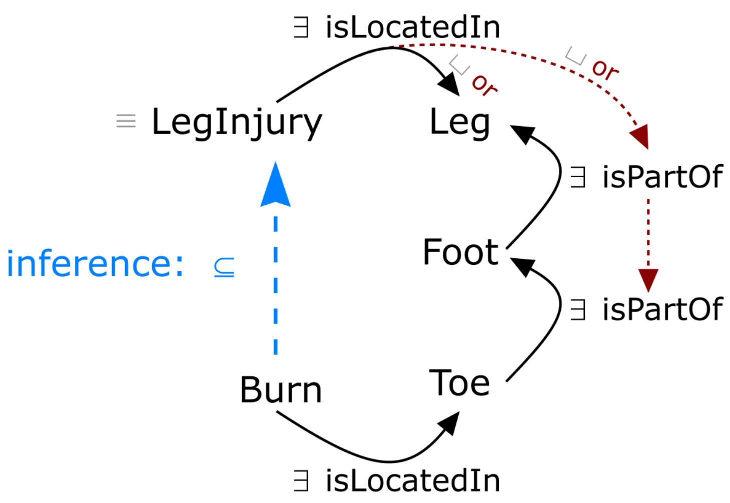


Fig. 8.15. Adapted SEP triples in action

Constraints and assumptions

Since the ontology used in our evaluation (see section 8.7) is normalised [31], only defined classes lead to inferences of new subsumption relations. Because of this only these defined classes need to be modified in order to create adapted SEP triples. However, in arbitrary ontologies SEP triples need to be applied to all classes in order to achieve a logically complete solution.

Transformations

Figure 8.16 shows the transformations that need to be applied to the defined classes in a knowledge base in order to create adapted SEP triples.

for: $R \circ S \dot{\sqsubseteq} R$ *(where S is a transitive property)*

$$\exists R \,.\, C \Rightarrow \exists R \,.\, (C \sqcup \exists S \,.\, C)$$

$$\forall R \,.\, C \Rightarrow \forall R \,.\, (C \sqcup \exists S \,.\, C)$$

$$\exists R^{-} \,.\, C \Rightarrow (\exists R^{-} \,.\, C) \sqcup (\exists S^{-} \,.\, (\exists R^{-} \,.\, C))$$

$$\forall R^{-} \,.\, C \Rightarrow \begin{array}{l} \forall R^{-} \,.\, C \\ \exists S^{-} \,.\, \top \sqsubseteq \forall R^{-} \,.\, C \end{array}$$

Fig. 8.16. Transformations for creating adapted SEP triples

The last transformation rule requires some explanation: a class transformed in this way captures all classes that match the basic inverse restriction, while also being restricted to some other class in the ontology ($\top$) via the secondary property (S), where that other class must also have the same basic restriction as its superclass.

Handling multiple transitive propagations

Chains of defined classes require special consideration as the rules in Figure 8.16 must be applied recursively. That is: when one defined class references another defined class, the transformed SEP triple restriction no longer matches the original definition. The original defined class must therefore be transformed to match the newly transformed definition of the second defined class. Chains of defined classes do not classify correctly without this additional transformation.

Figure 8.18 gives an example of such a case: suppose we take the ontology from Figure 8.15 and add a second SEP triple definition. If only the necessary and sufficient condition on the *FootComponent* class ($\equiv \exists\, isPartOf . (Foot \sqcup \exists\, isMultipleOf . Foot)$) is asserted, then the link from the *FootComponent* to the original *Foot* concept does not hold and the SEP triple inference cannot take effect; i.e. the *Burn2*

$$isLocatedIn \in R$$
$$isPartOf \in R_+$$
$$isMultipleOf \in R_+$$
$$LegInjury \equiv \exists\, isLocatedIn . \left(\begin{matrix} Leg \sqcup \\ \exists\, isPartOf . \left(\begin{matrix} Leg \sqcup \\ \exists\, isMultipleOf . Leg \end{matrix} \right) \end{matrix} \right)$$

Fig. 8.17. New query for adapted SEP triples with chained definitions

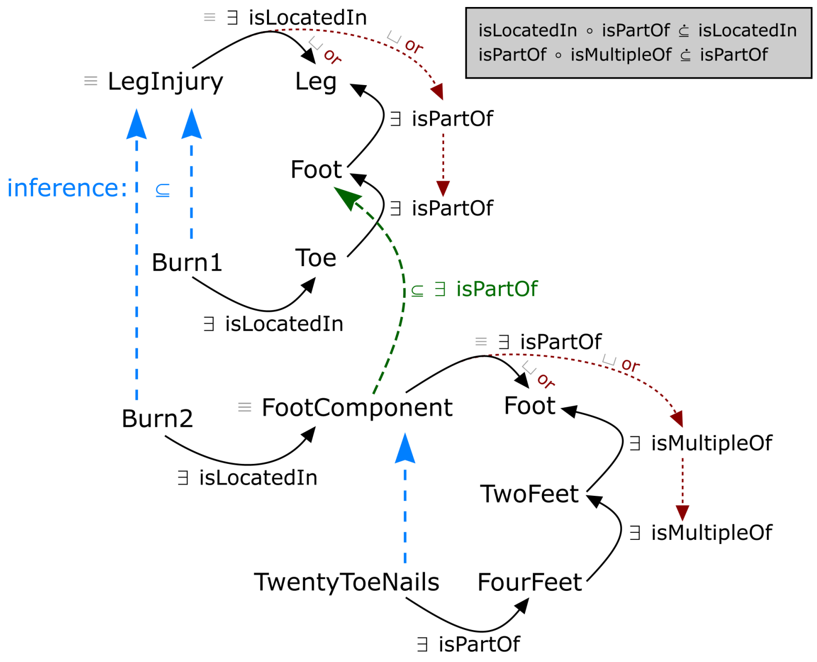


Fig. 8.18. Example of multiple transitive propagations

concept is not classified correctly. However, if an additional transformation is applied to *LegInjury*, resulting in the new definition of that class as shown in Figure 8.17, then the link indicated by the striped curved upwards pointing arrow is captured and the correct inference results. That is: *Burn2* is inferred as being a kind of *LegInjury*.

It should be noted that these kinds of chained universal restrictions may not need to be taken into account when creating SEP triples, depending on the ontology in question. However, some medical ontologies (such as GALEN) contain a substantial amounts of transitive propagation. A correct implementation is crucial in these cases.

Rector neglects to mention the need for a recursive algorithm when originally describing adapted SEP triples [29].

Discussion

Advantages of this modelling methodology are that it is logically correct and therefore, unlike the property subsumption method, will not produce any unexpected behaviour. It can also be applied selectively (unlike the potential implementation in the $\mathcal{SROIQ}$ description logic [14] based on complex property chain inclusion axioms [17]), so some concepts in an ontology can use transitive propagation, while others do not.

However, adapted SEP triples (unlike property subsumption) modify concept semantic (though not as drastically as classic SEP triples do) and may therefore be more difficult for a beginner to comprehend. They also require a somewhat complicated recursive transformation algorithm when dealing with chained transitive propagation.

8.6 Evaluation of Segmentation

The performance of this methodology was evaluated by various statistical measures.

(Tests were carried out on a 2.8 Ghz Pentium 4 with 2.5 GB of RAM running Racer 1.8.0 on Windows XP service pack 2.)

8.6.1 Segmentation Speed

Figure 8.19 gives a breakdown of how long various aspects of the segmentation process take. The first step is loading the target ontology. The next involves an initial pass over the ontology, scanning for and marking the classes to include in the eventual segment extraction process. Extraction constructs a new, self-contained ontology segment, which is then saved to disk.

As can be seen from the figure, the complete segmentation process takes an average of one minute to complete. However, most time is spent loading the ontology. Once the target ontology is in memory, the segmentation itself takes only around six seconds to complete. It can be observe that segments from large ontologies can be created with good computational efficiency, though this is, of course, dependent on the specific implementation of the extraction algorithm.

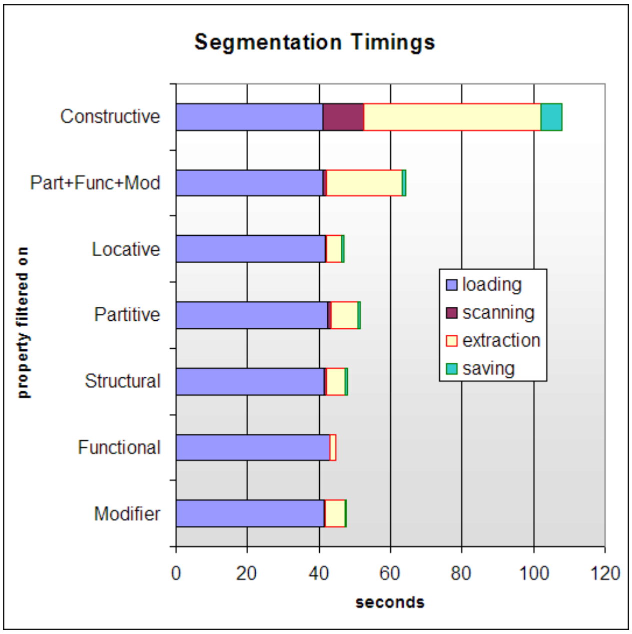


Fig. 8.19. Time to compute a segment

Performance is currently not fast enough for real-time user queries. However, the results show good potential for future optimisations, especially if loading times can be reduced by streaming segmentation techniques and/or caching. Furthermore, segmentation is not meant to replace querying. Instead, it enables efficient querying of otherwise intractable ontologies.

8.6.2 Basic Segmentation

The basic segmentation algorithm targeted around the GALEN "Heart" concept produced the results shown in Table 8.1. As can be seen from the table, the segment is roughly a quarter the size of the original ontology, with the number of properties being reduced the least and the number of primitive classes being reduced the most. A similar pattern can be observed when segmenting using different target classes.

Table 8.1. Basic segment of the *Heart* concept

	original	segment	size difference
number of classes	23139	5794	25%
primitive classes	13168	2771	21%
defined classes	9971	3023	30%
number of properties	522	380	71%
filesize in KB	22022	5815	26%

This reduction in size is not enough to enable classification given current memory and reasoner applications. All current tableaux algorithm-based description logic reasoner systems face a stack-overflow when attempting to classify the basic extract of GALEN. The filtering and boundary extraction algorithms do however create classifiable ontology segments (see Section 8.6.3).

8.6.3 Property Filtering Segmentation Results

A segment was produced by including only properties from each of the main property categories identified in Section 8.4.1. Segments using combinations of property categories were also produced. It was found that the combination of *Partitive*, *Functional* and *Modifier* properties produced the largest ontology that could still be classified successfully. Statistics for this combination segment are therefore also included in the tables below.

Table 8.2. Filtering segmentation size results

filter	total classes	defined classes	number of properties	size in KB
Modifier	99	10	56	63
Functional	129	17	22	57
Structural	357	29	74	258
Partitive	518	175	62	362
Locative	524	131	112	295
Part+Func+Mod	909	285	164	664
Constructive	5567	2954	284	5096
Basic seg.	5794	3023	380	5815
Original	23139	9971	522	22022

Table 8.2 gives an overview of the size of various property filtered segments. As can be seen from the results, segments could be reduced in size by an average factor of 20 over the original ontology and by a factor of five over the basic extraction methodology.

Probe classes

$ProbeHeart \equiv \exists\ attribute\,.\,Heart$

Fig. 8.20. Probe class use to test classification performance

The test query (*probe class*) in Figure 8.20 was introduced into every segmented ontology to test its classification performance. An approximate measure of the degree of knowledge present in a segment may be obtained by counting the number of new classes inferred as subclasses of the probe. The probe effectively answers the query "everything related to the Heart" by using the "*attribute*" property, which is the top-level property in GALEN.

Classification tests

Table 8.3 shows several classification statistics.

Note: The "new inferences" column only lists new inferences under the *probe class*. Many other new subclass relationships are inferred in each segment, but these are not necessarily relevant to the extract's target concept and were therefore not counted as part of this evaluation.

Table 8.3. Basic segment of the *Heart* concept

filter	defined classes	new inf.	speed in sec	ms per def.	new inf. per def.
Structural	29	1	5	172	0.03
Modifier	10	1	1	100	0.1
Locative	131	30	7	52	0.23
Part+Func+Mod	285	85	22	77	0.30
Partitive	175	58	11	63	0.33
Functional	17	13	2	118	0.76
Constructive	2162	n/a	n/a	n/a	n/a

Discussion

- The segment using all *Constructive* properties (combination of *Structural*, *Locative*, *Functional* and *Partitive* properties) was too large to classify.
- The *Functional* and *Partitive* segments produced the most new inferences relative to their size. This indicates that a majority of the knowledge in GALEN is covered by these two structures.
- As expected, the *Modifier* properties do not add very much complexity to a segment and are therefore almost always safe to include in any segment.
- *Structural* properties do not play a major role in the ontology, since they do not add much information.
- *Locative* properties are of small, but not insignificant consequence to the classification. This indicates that complexity of the anatomical model in GALEN is far greater than the complexity of disease model.

8.6.4 Boundary Class Segmentation Results

Boundary size results

As one might expect, the boundary extraction algorithm produces progressively smaller segments, in proportion with the boundary cut-off (note: we presume that the

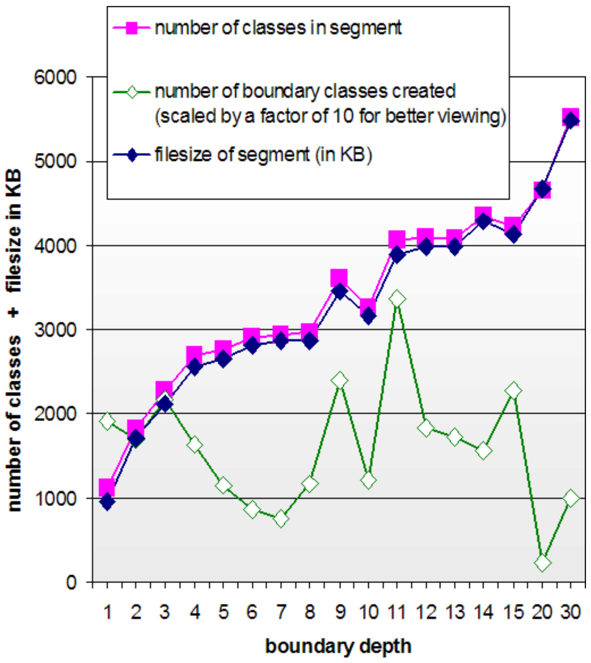


Fig. 8.21. Boundary depth, boundary classes and segment size

two instances of larger segments resulting from larger boundary depths shown in the Figure 8.21 is due to the depth-first implementation of the segmentation algorithm. So, a lesser boundary-depth could result in more of the ontology being selected, as the algorithm follows a different path through the link structure of the ontology). However, there seems to be no correlation between the number of boundary classes created at each cut-off level and the size of the resultant ontology. Figure 8.21 illustrates the differences in boundary sizes.

This result indicates that the link structure of the GALEN ontology is very interwoven and unpredictable. There seem to be no tight group of boundary classes that limit a particular extract and therefore also no way to cleanly divide an ontology into modules. That is, the complex ontological relationships cannot be cleanly divided into fixed categories. We should therefore expect traditional partitioning methodologies, such as those discussed in Section 8.8, to be of limited use in this domain.

Boundary classification results

Table 8.4 shows the results of the boundary classification testing. Only boundary depth "one" could be successfully classified.

Table 8.4. Boundary extract classification tests

boundary depth	defined classes	new inf.	speed in sec	ms per def.	new inf. per def.
1	279	2	34	121	0.007

Boundary extraction by itself provides a very good means of controlling the size of an extract, but does not seem to provide much optimization for classification. A combination of boundary extraction and filtering segmentation allows one to control both the classifiability and size of a segment. This combination represents the optimal segmentation strategy.

8.7 Evaluation of Transformation during Segmentation

8.7.1 Test Setup

Classification speed tests were carried out using the RACER 1.8 description logic reasoning system [11] on a 2.8 Ghz Pentium 4 with 2.5 GB of RAM running Windows XP. All tests were carried out utilizing the maximum memory possible in 32-bit Java applications running in Windows XP (1.5 GB). The figures quoted are the times spent in actual reasoning. Data transfer latency is not shown. Tests were run for as long as necessary. That is, classification failure is reported only if the reasoner application crashed while attempting to classify a particular ontology.

8.7.2 Ontology Segment Test Sets

The ontology segmentation algorithm described in in section 8.3 was therefore used to create a test set of smaller, classifiable segments of the complete GALEN ontology. This test set consisted of a total of 162 ontology extracts centred around the *Heart* concept from the GALEN ontology [32]. Segments were chosen so all the base-case extracts were tractable.

The GALEN ontology was filtered (using the algorithm described in section 8.4.1) using four individual meta-properties (locative, structural, partitive, functional) as well as four combinations of meta-properties (functional + modifier, structural + modifier, partitive + functional + locative, structural + functional). These property sets were used to generate a various ontology segments. Additionally, the depth of the link traversal algorithm was limited in order to produce even more tightly constrained versions of these ontologies. Segments were created with maximal depths for the recursive ontology segmentation algorithm ranging from one to five, as well as without any depth limit. Finally, different styles of transitive propagation, based upon rules harvested from the original GALEN ontology, were applied to each extract (no transitive propagation, property subsumption and adapted SEP triples). This lead to a total of 144 test ontologies ($(4 + 4) \times 6 \times 3 = 144$).

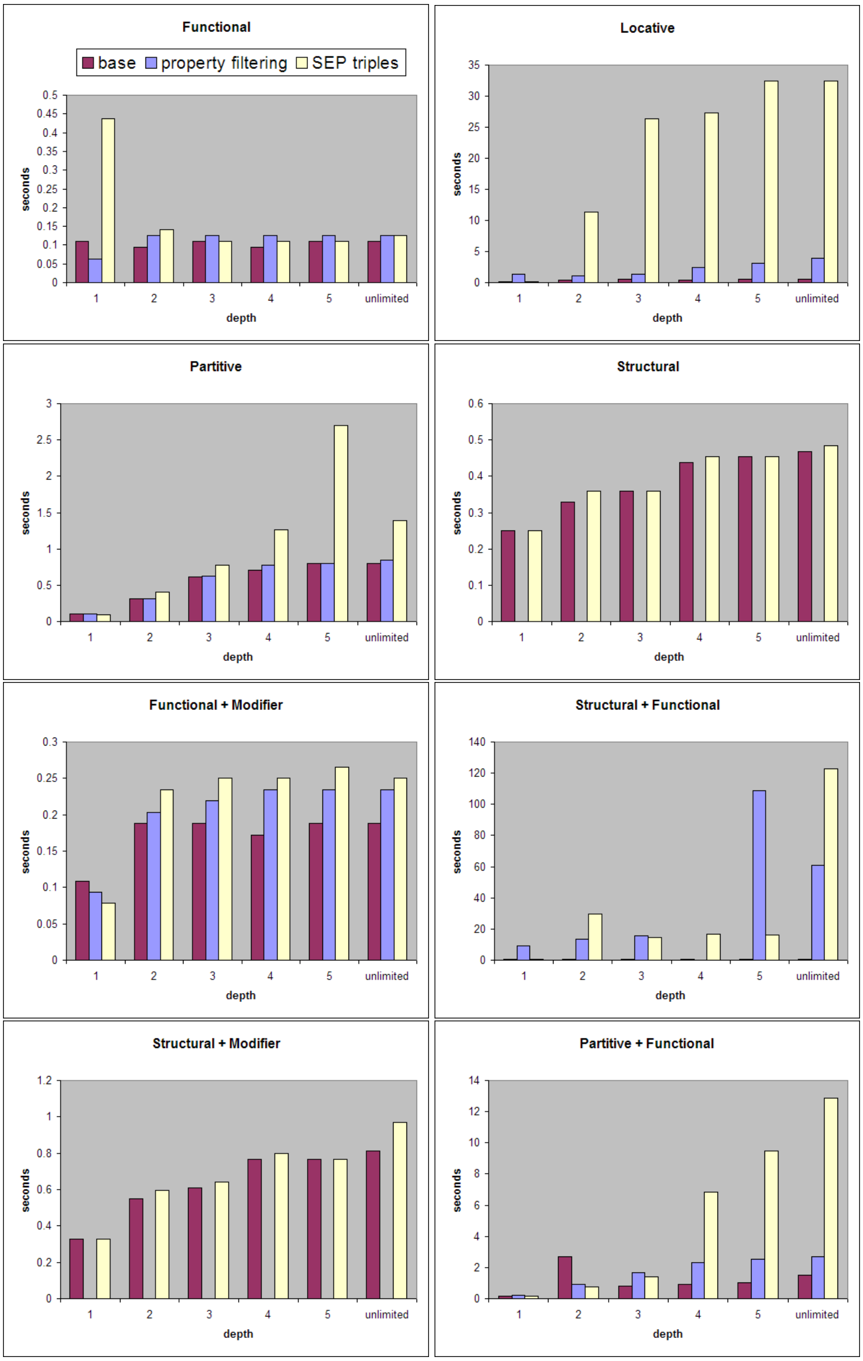


Fig. 8.22. Timing test results

8.7.3 Test Evaluation

The following observations can be made from the tests shown in Figure 8.22:

- Classification performance for *functional* properties is similar regardless of the method used. This is due to the relatively small amount of transitive propagation in those extracts.
- Extracts transformed to employ SEP triples using *locative* properties take an order of magnitude longer to classify than those using *partitive* segments. This is in spite of the *partitive* extracts having more actual SEP triples (388 vs. 244 triples in the case of unlimited extract depth). One can therefore conclude that classification speed is not directly correlated with the number of triples, but is more complex of an issue. Indeed, in both cases, the property subsumption technique performs very well.
- All the *structural* segments that employ property subsumption transformations crash the reasoner.
- The slowest classification performance in the test set results from an extract combining the *structural* and *functional* properties. Both of these property sets can be classified individually within about a second. However, the combination performs up to one hundred times slower. A similar pattern can be observed from the combination of *partitive* and *functional* properties.
- Extracts filtered using *structural* properties are unclassifiable when using the property subsumption technique. However when *functional* properties and *structural* properties are combined, this combination extract suddenly becomes tractable. However, classification performance suffers by almost three orders of magnitude compared to *functional* properties on their own.

In summary: *structual* properties scale extremely badly for property subsumption. *Locative* properties scale badly for SEP triples. SEP triples performance is slower than property sumbsumption and far slower than classification without transitive propagation. In rare cases, adding more information/complexity causes previously intractable knowledge bases to become classifiable.

8.7.4 Discussion of Transformation Results

The property subsumption method of adding transitive propagation to an ontology results in a tangled property hierarchy, which can create complex cycles. These cycles can make classification completely intractable. This technique is also logically inaccurate and can result in incorrect/unintended inferences. However, performance is only slightly worse than the base-case.

Adapted SEP triples, on the other hand, add a large number of disjunctions in the knowledge base, thereby increasing the number of possibilities a tableaux reasoning system must explore in order to classify the ontology [13]. This can result in a significant increase in classification time. However, unlike the property subsumption mechanism, the increase in complexity does not result in intractability and is logically correct.

8.8 Related Work

8.8.1 Overview

The idea of extracting a subset of a larger ontology is referred to by many different names by different authors. Research regarding views, segments, extracts, islands, modules, packages and partitions may be broken down into three main categories:

1. **Query-based methods**
2. **Network partitioning**
3. **Extraction by traversal**

The research presented here falls into category three. These are first suggested in [40] and further refined in [8].

8.8.2 Query-Based Methods

Many researchers, taking inspiration from the databases field, define ontological queries in an SQL-like syntax. As illustrated in figure 8.23, these queries can return sub-ontology segments as their answer-sets.

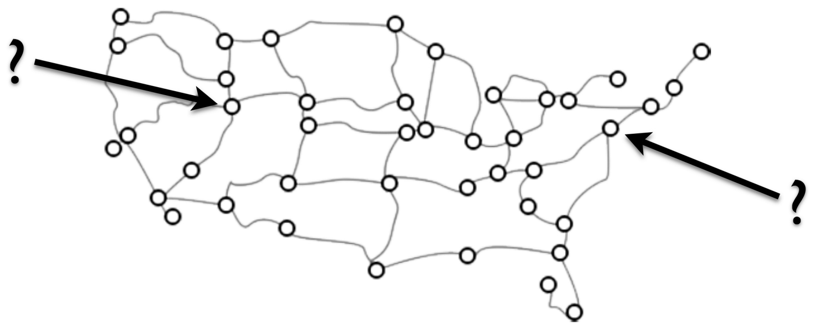


Fig. 8.23. Segmentation by querying

SparQL

The SPARQL query language [38] defines a simple query mechanism for RDF. Multiple queries are required in order to extract complex knowledge as, for example, a class and its *transitive closure* (all classes related to it). SparQL might be a good low-level tool for implementing ontology segmentation, but is not a solution in and of itself.

KAON views

Volz and colleagues define an ontology view mechanism based upon the RQL query language [48]. They highlight RQL [1] as the only RDF query language that takes the semantics of RDF Schema into account. Their view system has the ability to place each concept in its corresponding place in the complete RDF hierarchy. This practice, similar to the algorithm presented in Section 8.3, gives a more complete picture of the structure of a query answer than merely returning the relevant concepts in isolation. They do not however provide a means of *materializing* a view, i.e. views are transient: they are discarded as soon as they have served their purpose.

RVL

Magkanaraki and colleagues present a similar approach to Volz's, except their system also allows queries to reorganize the RDFS hierarchy when creating a view [21]. This allows views to be customized on-the-fly for specific applications' requirements. They however also side-step the ontology updating problem by only creating *virtual views*. Their views are merely a collection of pointers to the actual concepts, and are discarded after they have served their purpose.

Discussion

Query-based methods provide a view mechanism similar to SQL. This makes them intuitively familiar to computer scientists with a background in databases. The shortcomings of these approaches are that they provide only very low-level access to the semantics of the ontology being queried. They also do not address the issue of updating the original ontology when an extract is changed, although they would be a in unique position to offer such a feature. Query-based views are good for getting very small, controlled, single-use extracts, which are tightly focused around a few concepts of interest.

By contrast, the methods presented herein create self-standing, persistent, multi-use ontology segments. That is, the segments have a life of their own: they can be transformed, updated, shared, annotated, plugged into applications and otherwise manipulated in myriad of ways.

8.8.3 Network Partitioning

The basic idea of partitioning comes from Herbert Simon. He asserts that any system has the property of *near-complete decomposability* [41]. That is, we can always find clusters of objects that are more related to each other than to the other objects around them. How complete a decomposition is possible depends on the nature of the system in question.

Researchers in networking use algorithms to organize the nodes on a network into inter-related *islands* [2]. Some ontology researchers propose applying a similar methodology to segmenting ontologies (as illustrated by figure 8.24).

An ontology can, from this point of view, be viewed as a network of nodes connected by links. The class hierarchy can be interpreted as a directed acyclic graph (DAG) and any relations between classes can be represented as links between the nodes (a simplified model of the paradigm presented in Section 8.2.1).

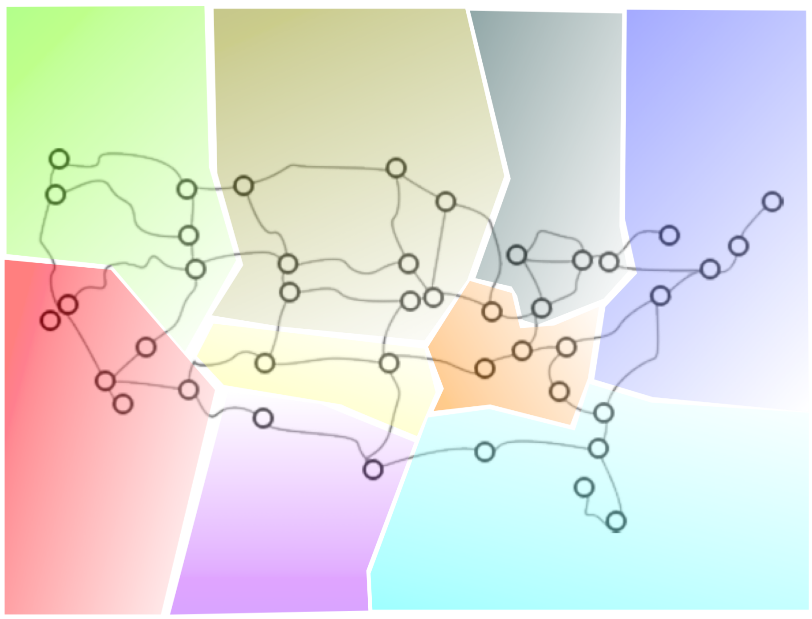


Fig. 8.24. Segmentation by partitioning

Structure-based partitioning

Stuckenschmidt and Klein present a method of partitioning the classes hierarchy into modules [44]. They exploit the structure of the hierarchy and constraints on properties' domains and ranges (for example: the "hasGender" property might have a domain of "Animal" and a range of "Male or Female") to iteratively break the ontology up into dynamically sized modules. This method does not take OWL restrictions, which can act as additional links between concepts, into account. Instead it relies on the globally asserted domain & range constraints. However, domains and ranges are optional and may not therefore be asserted in a given ontology.

Structure-based partitioning is primarily meant for breaking an ontology into broad packages or modules so that it can be more easily maintained, published and/or validated. However, this process destroys the original ontology, leaving it decomposed into whatever modules the partitioning algorithm deemed appropriate. Moreover, ontologies, particularly those modeled in OWL, tend to be more semantically rich than a simple network abstraction will capture.

SNARK and Vampire

MacCartney et al. use the same partitioning idea to solve a different problem: they present a first-order logic theorem prover (SNARK) [20], which decomposes the knowledge base into self-contained mini-prover partitions, which then communicate with each other using message passing techniques. The researchers thereby successfully improve the efficiency of their reasoning algorithm when answering queries over large knowledge bases.

Tsarkov and Horrocks [46] use a similar approach for optimizing the classification performance of the Vampire first-order logic theorem prover [33] when classifying description logic ontologies.

8.8.4 Extraction by Traversal

Ontology extraction by traversal (as illustrated by figure 8.25), similar to the network partitioning approach, also sees the ontology as a networking or graph. However, instead of decomposing the entire graph into modules, this methodology starts at a particular node (usually a concept) and follows its links, thereby building up a list of nodes (concepts) to extract. A key difference is that this leaves the structure of the original ontology intact: it creates an extract, not a decomposition.

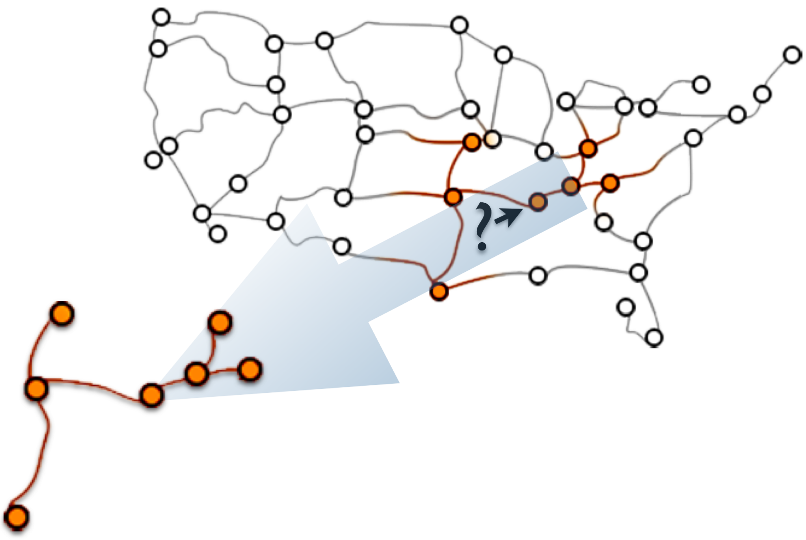


Fig. 8.25. Segmentation by traversal

Extracting Modules from Ontologies

Grau and colleagues [10] present a method for modularizing OWL ontologies into logically independent modules. Each module contains all the axioms relevant to the meaning of the concepts in that module. More densely interconnected ontologies (such as GALEN) result in several very large modules, while simpler ontology structures produce much smaller modules. This approach does not however allow the user to constrain module size (e.g by property filtering or boundary limiting), since modules are guaranteed to contain all relevant axioms and size constraining techniques might break this warrant.

This module extraction approach decomposes ontologies into distinct modules, yielding, from an external perspectives, structures similar to other network partitioning approaches. However, the approach internally uses a more traversal-like procedure.

PROMPT

Noy and Musen present an extension to the PROMPT suite [23] of ontology maintenance tools, which are themselves plug-ins to the Protégé ontology editor [25]. Their extraction methodology [24] focuses on *traversal directives*, which define how the ontology links should be traversed. Collections of directives completely and unambiguously define an ontology view and can themselves be stored as an ontology. They also introduce the concept of *boundary classes* around the edges of an extract. However, their view of boundary classes differs from the perspective given in section 8.4.2.

Noy's research establishes the mechanics of ontology view extraction, but does not address how her system might be used to construct relevant, useful and computationally tractable segments.

MOVE

Bhatt, Wouters and company have a different focus: They present the Materialized Ontology View Extractor (MOVE) system for distributed sub-ontology extraction [3]. It is a generic system that can theoretically be adapted to work with any ontology format. The system extracts a sub-ontology based on a user's labelling of which ontology terms to include and which to exclude. It also has the ability to optimise an extract based upon a set of user selectable optimisation schemes. These schemes can produce either the smallest possible extract, a medium size one, or include as much detail as possible. These extracts can be further restricted by enforcing a set of additional constraints. Their system can, for example, enforce the semantic completeness and well-formedness of an extract [50].

However, the primary focus of Bhatt and Wouters' architecture is parallel processing. While, their extract system performs very poorly when run on a single machine (17 minutes to produce an extract from a 5000-concept ontology), it achieves optimum performance using around five separate processors.

We argue that speed is not a critical factor in the extraction process. Performance is too poor to facilitate an instant, on-demand extraction web-service, but not poor enough that it becomes a serious problem. For example, extraction tests on the GALEN ontology by these authors took in the order of two to five minutes to complete.

Discussion

Both MOVE and PROMPT produce a materialized view, i.e. a self-standing ontology that has no direct connection with its origin. Both also have the notion of the transitive closure of a concept (Wouters et al. call this *semantic completeness* [50]). However, neither methodology addresses the important issue of how to update the main ontology if the view is modified, how to transform the ontology on-the-fly while extracting, nor do they discuss the ability to classify an extract using description-logic reasoning systems. Finally, neither systems make use of meta-information about an ontology's semantics in determining the best extract. The user must make a great deal of manual selections and choices for each new extract he or she wishes to produce.

By contrast, the segmentation algorithms presented herein automates the extraction process as much as possible by taking advantage of meta-information. Additionally, these methods have the ability to transform the extracted ontology segments (see Section 8.4.1).

The key difference between the approach present here and the other approaches is that we do not aim to create views of one type or another. Instead, we aim to produce independently useful and usable ontologies.

8.9 Conclusion

Ontologies with over ten-thousand classes suffer severely from scaling problem. Segmentation by traversal is a way of overcoming these difficulties. Developers can use ontology segmentation techniques to quickly and easily create the relevant, self-standing custom ontologies they require, instead of having to rely on the initial authors' decomposition. Developers can also transform segments while they are being extracted using different ontology design patterns. Ontology segments can be specialized further by only including links of specific types in the extract (property filtering), limiting the depth of the link traversal algorithm (boundary extraction), or a combination of both.

The methods presented take advantage of many ontology maintenance principles: normalisation [31], upper-ontologies [28] and rich property hierarchies [30] are all taken into account in order to produce more relevant segments that are easy to transform.

Evaluation has shown that segmenting ontologies can decrease their size considerably and significantly improve their performance. The size of the GALEN ontology was reduced by a factor of 20. The classification performance of transitive propagation could be customized to either be very fast when using the property subsumption approximation, or relatively slow but reliable using adapted SEP triples. Depending on choice of segment target, filtering technique, boundary limit and ontology transformation, classification performance could be varied significantly. Classification speeds ranged from fractions of seconds to classification failure. All these results yield useful insights into an ontology's meta-structure.

A application for generating custom ontology segments is available for free download at: `http://www.co-ode.org/galen`

References

1. Alexaki, S., Christophides, V., Karvounarakis, G., Plexousakis, D., Tolle, K., Amann, B., Fundulaki, I., Scholl, M., Vercoustre, A.-M.: Managing RDF Metadata for Community Webs. In: ER 2000: Proceedings of the Workshops on Conceptual Modeling Approaches for E-Business and The World Wide Web and Conceptual Modeling, pp. 140–151 (2000)
2. Batagelj, V.: Analysis of large network islands. Dagstuhl Semina 03361, University of Ljubljana, Slovenia, Algorithmic Aspects of Large and Complex Networks (August 2003)
3. Bhatt, M., Wouters, C., Flahive, A., Rahayu, W., Taniar, D.: Semantic completeness in sub-ontology extraction using distributed methods. In: Laganá, A., Gavrilova, M.L., Kumar, V., Mun, Y., Tan, C.J.K., Gervasi, O. (eds.) ICCSA 2004. LNCS, vol. 3045, pp. 508–517. Springer, Heidelberg (2004)
4. Bray, T.: What is RDF? `http://www.xml.com/pub/a/2001/01/24/rdf.html` (January 2001)
5. Brin, S., Page, L.: The anatomy of a large-scale hypertextual Web search engine. Computer Networks and ISDN Systems 30(1–7), 107–117 (1998)
6. Dameron, O.: JOT: a Scripting Environment for Creating and Managing Ontologies. In: 7th International Protégé Conference (2004)
7. Drummond, N., Horridge, M., Wang, H., Rogers, J., Knublauch, H., Stevens, R., Wroe, C., Rector, A.: Designing User Interfaces to Minimise Common Errors in Ontology Development: the CO-ODE and HyOntUse Projects. In: Cox, S.J. (ed.) Proceedings of the UK e-Science All Hands Meeting (September 2004)
8. dquin, M., Sabou, M., Motta, E.: Modularization: a key for the dynamic selection of relevant knowledge components. In: First International Workshop on Modular Ontologies (WOMO) (2006)
9. Golbeck, J., Fragoso, G., Hartel, F., Hendler, J., Oberthaler, J., Parsia, B.: National Cancer Institute's Thésaurus and Ontology. Journal of Web Semantics (2003)
10. Grau, B.C., Horrocks, I., Kazakov, Y., Sattler, U.: Just the right amount: Extracting modules from ontologies. In: Proc. of the 16th International World Wide Web Conference (2007)
11. Haarslev, V., Möller, R.: RACER System Description. In: Goré, R.P., Leitsch, A., Nipkow, T. (eds.) IJCAR 2001. LNCS, vol. 2083, p. 364. Springer, Heidelberg (2001)
12. Hendler, J.: On beyond ontology. Keynote talk. In: International Semantic Web Conference (2003)
13. Horrocks, I.: Optimisation techniques for expressive description logics. Technical Report UMCS-97-2-1, University of Manchester, Department of Computer Science (February 1997)
14. Horrocks, I., Kutz, O., Sattler, U.: The Even More Irresistible SROIQ. Technical report, University of Manchester (2005)
15. Horrocks, I., Patel-Schneider, P.F., van Harmelen, F.: From SHIQ and RDF to OWL: The making of a web ontology language. Journal of Web Semantics 1, 7–26 (2003)
16. Horrocks, I., Rector, A.L., Goble, C.A.: A Description Logic Based Schema for the Classification of Medical Data. In: KRDB (1996)
17. Horrocks, I., Sattler, U.: Decidability of $\mathcal{SHIQ}$ with complex role inclusion axioms. Artificial Intelligence 160(1–2), 79–104 (2004)
18. Knublauch, H., Fergerson, R.W., Noy, N., Musen, M.A.: The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 229–243. Springer, Heidelberg (2004)

19. Lutz, C., Sattler, U., Tendera, L.: The complexity of finite model reasoning in description logics. In: Baader, F. (ed.) CADE 2003. LNCS, vol. 2741, pp. 60–74. Springer, Heidelberg (2003)
20. MacCartney, B., McIlraith, S., Amir, E., Uribe, T.E.: Practical Partition-Based Theorem Proving for Large Knowledge Bases. In: Proceedings of the Nineteenth International Conference on Artificial Intelligence (IJCAI 2003), pp. 89–96 (2003)
21. Magkanaraki, A., Tannen, V., Christophides, V., Plexousakis, D.: Viewing the Semantic Web through RVL Lenses. Journal of Web Semantics 1(4), 29 (2004)
22. McGuinness, D.L., van Harmelen, F.: OWL Web Ontology Language Overview, W3C Recommendation (February 2004)
23. Noy, N., Musen, M.A.: The PROMPT Suite: Interactive Tools For Ontology Merging And Mapping. International Journal of Human-Computer Studies 59(6), 983–1024 (2003)
24. Noy, N., Musen, M.A.: Specifying ontology views by traversal. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 713–725. Springer, Heidelberg (2004)
25. Noy, N.F., Fergerson, R.W., Musen, M.A.: The Knowledge Model of Protégé-2000: Combining Interoperability and Flexibility. In: Dieng, R., Corby, O. (eds.) EKAW 2000. LNCS, vol. 1937, pp. 17–32. Springer, Heidelberg (2000)
26. Odell, J.J.: Six different kinds of composition. Journal of Object-Oriented Programming 5(8), 10–15 (1994)
27. Parsia, B., Sirin, E.: Pellet: An OWL DL reasoner. In: ISWC 2004 (2004)
28. Pease, A., Niles, I., Li, J.: The suggested upper merged ontology: A large ontology for the semantic web and its applications. In: Working Notes of the AAAI 2002 Workshop on Ontologies and the Semantic Web, July 28-August 1 (2002)
29. Rector, A.: Analysis of propagation along transitive roles: Formalisation of the GALEN experience with medical ontologies. In: DL 2002 (2002)
30. Rector, A., Rogers, J.: Ontological Issues in using a Description Logic to Represent Medical Concepts: Experience from GALEN. In: IMIA WG6 Workshop (1999)
31. Rector, A.L.: Normalisation of ontology implementations: Towards modularity, re-use, and maintainability. In: EKAW Workshop on Ontologies for Multiagent Systems (2002)
32. Rector, A.L., Bechhofer, S., Goble, C., Horrocks, I., Nowlan, W.A., Solomon, W.D.: The GRAIL concept modelling language for medical terminology. Artificial Intelligence in Medicine 9(2), 139–171 (1997)
33. Riazanov, A., Voronkov, A.: Vampire 1.1 (system description). In: Goré, R.P., Leitsch, A., Nipkow, T. (eds.) IJCAR 2001. LNCS(LNAI), vol. 2083, pp. 376–380. Springer, Heidelberg (2001)
34. Rogers, J.: OpenGALEN: Making the impossible very difficult, `http://www.opengalen.org/` (August 2005)
35. Rogers, J., Rector, A.: GALEN's model of parts and wholes: Experience and comparisons. In: Proceedings of AMIA Symposium, pp. 714–718 (2000)
36. Rosse, C., Mejino, J.L.V.: A reference ontology for biomedical informatics: the Foundational Model of Anatomy. Journal of Biomedical Informatics 36(6), 478–500 (2003)
37. Schulz, S., Hahn, U., Romacher, M.: Part-Whole Reasoning in Medical Ontologies Revisited: Introducing SEP Triplets into Classification-Based Description Logics. In: AMIA Annual Fall Symposium, November 1998, pp. 830–834. Hanley & Belfus (1998)
38. Seaborne, A., Prud'hommeaux, E.: SparQL Query Language for RDF, `http://www.w3.org/TR/rdf-sparql-query/` (February 2005)
39. Seidenberg, J., Rector, A.: Representing Transitive Propagation in OWL. In: Embley, D.W., Olivé, A., Ram, S. (eds.) ER 2006. LNCS, vol. 4215, pp. 255–266. Springer, Heidelberg (2006)

40. Seidenberg, J., Rector, A.: Web ontology segmentation: Analysis, classification and use. In: 15th International World Wide Web Conference (May 2006)
41. Simon, H.A.: The Sciences of the Artificial, ch. 7, pp. 209–217. MIT Press, Cambridge (1969)
42. Smith, B., Williams, J., Schulze-Kremer, S.: The Ontology of the Gene Ontology. In: Proceedings of AMIA Symposium (2003)
43. Stearns, M.Q., Price, C., Spackman, K.A., Wang, A.Y.: SNOMED clinical terms: overview of the development process and project status. In: Proceedings of AMIA Symposium, pp. 662–666 (2001)
44. Stuckenschmidt, H., Klein, M.: Structure-Based Partitioning of Large Class Hierarchies. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 289–303. Springer, Heidelberg (2004)
45. TheFreeDictionary.com. Transitivity definition (2004)
46. Tsarkov, D., Horrocks, I.: Dl reasoner vs. first-order prover. In: Description Logic Workshop (DL 2003). CEUR, vol. 81, pp. 152–159 (2003)
47. Tsarkov, D., Horrocks, I.: Reasoner prototype: Implementing new reasoner with datatypes support. WonderWeb Project Deliverable (2003)
48. Volz, R., Oberle, D., Studer, R.: Views for light-weight web ontologies. In: Proceedings of the ACM Symposium on Applied Computing (SAC)(2003)
49. Winston, M., Chaffin, R., Herrmann, D.: A taxonomy of part-whole relations. In: Cognitive Science, vol. 11, pp. 417–444 (1987)
50. Wouters, C., Dillon, T.S., Rahayu, J.W.J., Chang, E., Meersman, R.: Ontologies on the MOVE. In: Lee, Y., Li, J., Whang, K.-Y., Lee, D. (eds.) DASFAA 2004. LNCS, vol. 2973, pp. 812–823. Springer, Heidelberg (2004)

9

Traversing Ontologies to Extract Views

Natalya F. Noy and Mark A. Musen

Stanford Center for Biomedical Informatics Research, Stanford University, USA
{noy,musen}@stanford.edu

Summary. One of the original motivations for ontology research was the belief that ontologies can help with reuse in knowledge representation. However, many of the ontologies that are developed with reuse in mind, such as standard reference ontologies and controlled terminologies, are extremely large, while the users often need to reuse only a small part of these resources in their work. Specifying various views of an ontology enables users to limit the set of concepts that they see. In this chapter, we develop the concept of a *Traversal View*, a view where a user specifies the central concept or concepts of interest, the relationships to traverse to find other concepts to include in the view, and the depth of the traversal. For example, given a large ontology of anatomy, a user may use a Traversal View to extract a concept of `Lung` and organs and organ parts that surround the lung or are contained in the lung. We define the notion of Traversal Views formally, discuss their properties, present a strategy for maintaining the view through ontology evolution and describe our tool for defining and extracting Traversal Views.

9.1 Ontology Views

Ontologies constitute an integral and important part of the Semantic Web. For the Semantic Web to succeed, developers must create and integrate numerous ontologies, from general top-level ontologies, to domain-specific and task-specific ones. One of the original motivations behind ontology research was the belief that ontologies can help with reuse in knowledge representation [6]. By virtue of being formal and explicit representations of a domain, ontologies could represent shared domain descriptions that different applications and agents use. When a person developing a new application chooses to reuse a published shared ontology of the domain rather than to create his own model, he gains a number of advantages: not having to reinvent the wheel, using an ontology that has already been tested in other applications, and, perhaps, most important of all, tremendously facilitating the path to information integration among different applications that use the same ontology.

Currently, there are several efforts to develop standard reusable ontologies in various domains and to make them available on the Semantic Web: from the generic upper-level ontology of SUO[1] and lexical corpus of WordNet [4] to domain-specific

[1] `http://suo.ieee.org/`

ontologies such as UMLS [10]. Consider for example the Foundational Model of Anatomy (FMA)—a declarative representation of anatomy developed at the University of Washington [16]. The ontology represents the result of manual and disciplined modeling of the structural organization of the human body. While the FMA ontology is a relatively recent development, many in medical informatics already consider it to be a tremendous resource that will facilitate sharing of information among applications that use anatomy knowledge. However, the size and complexity of the ontology is immense: approximately 70,000 concepts at the time of this writing. As a result, the project authors often get requests for self-contained portions of the ontology that describe only specific organs and organ parts.

In general, while providing a shared, tested, and well-accepted vocabulary, many of the large standard resources pose a formidable challenge to Semantic Web users: these resources are huge, often containing tens of thousands of concepts. However, many Semantic Web users need only a small fraction of the resource for their application. Currently such a user still needs to make sense of the complete resource, importing it as a whole, and dragging along this extra "baggage" of concepts, most of which he is never going to use. In addition to the cognitive challenge of understanding the large resource (or at least figuring out which swaths of it are not relevant and can be safely ignored), there can be a considerable computational penalty as well if the user needs to perform reasoning with the whole ontology.

Therefore, users need the ability to extract self-contained portions of ontologies and to use these portions in their applications. For example, suppose a user is interested in applications related to lung cancer. This user may specify that he needs a portion of the FMA that contains everything that is directly related to the *lung* as well as definitions of all organs and organ parts that surround the lung. The user may also ask to include organs and organ parts that are "twice removed" from the lung—related to the lung through one other concept. Or, the user may ask for everything related to the lung concept—a transitive closure of all relations in which the concept participates.

The notion of creating a self-contained portion of a resource has long been an area of research in databases. A database *view* provides exactly that: users specify a query that extracts a portion of database instances satisfying the query, creating a specific *view* on the data in the database. Similarly, we call a portion of an ontology an **ontology view**.

In databases, a view is specified as a query: all instances satisfying the query constitute the view. Current research on ontology views (see Section 9.7) takes a similar approach: an ontology view is specified as a query in some ontology-query language. However, this query-based approach does not allow users to specify a portion of an ontology that results from a particular traversal of ontology links, as in the *lung* example above. Therefore, we suggest a complementary way of defining ontology views: by **traversal specification**. In such a specification, we define a starter concept or concepts, the "central" or "focus" concepts in the result, and the details of the relationships to traverse, starting at these concepts. We call such a view a **Traversal View**.

Another way of defining a view is by defining meta-information in the ontology itself that describes which parts of the ontology should appear in a particular view.

This meta-information includes annotation of concepts and relations in the ontology with information describing the perspectives in which these concepts and relations should appear, and how the terms should be presented or named in each perspective. For instance, while the FMA takes a structure-based view of anatomy and is developed as a general reference model, a radiologist or a surgeon may use different terms or view some relationships differently. Similarly, an ontology developer may want to indicate that certain concepts or relations should be displayed only to the users who identify themselves as experts, while presenting a simpler, trimmed-down view for novices. Having such meta-information in the ontology enables the generation of different views and perspectives automatically.

We believe that various types of view definitions—query-based and traversal-based, views based on meta-information, and perhaps several others—are necessary to provide full flexibility to Semantic Web users in extracting specific parts of an ontology. We are developing a suite of tools that provides a convenient interface to allow users to specify views in different ways, to examine the resulting views, and to use them in their applications. In this chapter, we focus on Traversal Views, defining their properties, describing an algorithm for computing them, and presenting our user-centered tools for specifying and using them.

More specifically, this chapter makes the following contributions:

- defines the approach of specifying ontology views through traversal of concepts
- presents alternative algorithms for finding the results of traversal-view definitions
- identifies properties of Traversal Views
- presents a strategy for maintaining Traversal Views through ontology evolution
- describes a user-oriented tool for specifying Traversal Views

9.2 Specification of Traversal Views

In this work, we mainly consider ontologies expressed in RDF Schema and therefore consisting of classes, their properties, and instances [19]. However, as we discuss in Section 9.3, our approach can be easily adapted to ontologies expressed in OWL. In fact, our implementation, which we describe in Section 9.6, works both with RDFS and OWL ontologies.

We combine all the properties related to a class or an instance in a **concept definition**:

Definition 1 (Concept definition). *Given an RDF graph G and an RDF resource R, the definition of R in the graph G, $Def(R, G)$, is a set of RDF triples in G such that:*

- *any RDF triple with R as a subject is in $Def(R, G)$*
- *for any property P such that R is in the domain of P (as defined by $rdfs$: $domain$), any triple with P as a subject is in $Def(R, G)$*

No other triples are in $Def(R, G)$. ◇

This definition implies for example, that a definition for a class C includes all the property range statements for all the properties that contain C in their domain. Similarly, they include all the triples specifying property values for C.

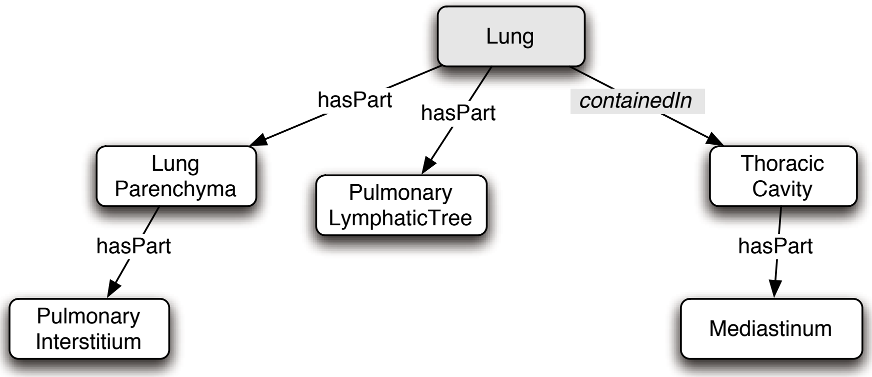


Fig. 9.1. An excerpt from the FMA, showing the class `Lung` and some of its relationships

We can now define the notion of Traversal Views. Traversal Views help users exploring a new ontology to find its subset that covers a particular "topic." For instance, one may not be interested in the entire human anatomy, but just in the lung and the organs that are physically close to it (say, separated from it by at most one other anatomical structure). Traversal Views enable users to extract a manageable and self-contained portion of the ontology relevant to their needs, allowing them to use and extend it for their own purposes.

Intuitively, we specify a Traversal View by specifying a starter concept (e.g., `Lung`), a list of relationships (property names) that should be traversed and the maximum distance to traverse along each of the relationships.

Example 1. Consider an excerpt from the FMA ontology presented in Figure 9.1. This excerpt contains the class `Lung`, some of its parts, the class it is contained in (`ThoracicCavity`), and its parts.[2] We can specify a traversal view that starts at the class `Lung` and traverses the `hasPart` relationship for 2 steps and the `containedIn` relationship for 1 step. We then compute the view in the following way:

1. `Lung` is included in the view
2. If `Lung` is in the domain of the property `hasPart`, then all classes in the range of this property are also included in the view. We will denote the set of these classes $C_{hasPart}$. From the classes in Figure 9.1, at this step, we will add `LungParenchyma` and `PulmonaryLymphaticTree` to the view and to $C_{hasPart}$.
3. If `Lung` was also an instance (RDF Schema does not prevent classes from being instances as well) and had a value for the property `hasPart`, those values are also included in the view. We also add these values to the set $C_{hasPart}$. The view now contains the traversal along the `hasPart` property of depth 1.

[2] We do not include all parts of `Lung` or any related classes for simplicity. In the FMA, all the classes in Figure 9.1 have many more parts.

4. We repeat steps 2 and 3 once for each concept in the set $C_{hasPart}$ to add values for the traversal along the `hasPart` property of depth 2. In the example, in Figure 9.1, this step will add the class `PulmonaryInterstitium` to the view.
5. We repeat steps 2 and 3 for the class `Lung` and property `containedIn` once, adding values for the traversal along the property `containedIn` of depth 1. This step will add the class `ThoracicCavity` from Figure 9.1 to the view.

As a result of this process, the view will contain all classes in Figure 9.1, except for `Mediastinum`. ◇

We now present a formal definition of a traversal view. The definition consists of two parts: the view specification (Definitions 2 and 3) and the view computation (Definitions 4 and 5) .

Definition 2 (Traversal Directive). *A* ***traversal directive*** *D for ontology O is a pair* $\langle C_{st}, \mathcal{PT} \rangle$ *where*

- C_{st} *is a class or an instance in O (the starter concept of the traversal);*
- $\mathcal{PT}$ *is a set of* ***property directives***. *Each property directive is a pair* $\langle P, n \rangle$*, where P is a property in O and* n *is a nonnegative integer or infinity (*inf*), which specifies the depth of the traversal along the property P. If* $n = \text{inf}$*, then the traversal includes a transitive closure for P starting with* C_{st}*.* ◇

In Example 1, we defined a single traversal directive, with `Lung` playing the role of the starter concept C_{st} and two property directives in $\mathcal{PT}$: $\langle hasPart, 2 \rangle$ and $\langle containedIn, 1 \rangle$.

Definition 3 (Traversal View Specification). *A* ***Traversal View specification*** *T is a set of traversal directives* $\mathcal{TD}$ ◇

The view specification in Example 1 contained only one traversal directive.

Example 2. Suppose that in addition to the concept `Lung` and its related concepts from the view in Example 1, we also want to include all parts of the lung in the view (without limiting the depth of the traversal). We will then include the following directive into the view specification (in addition to the directives in Example 1):

- $C_{st} = Lung$
- $\mathcal{PT} = \{\langle hasPart, \text{inf} \rangle\}$ ◇

We define the procedure for computing Traversal Views recursively as follows.

Definition 4 (Traversal Directive Result)
Given a traversal directive $D = \langle C_{st}, \mathcal{PT} \rangle$ *for an ontology O, a* ***traversal directive result*** $D(O)$ *(the result of applying directive D to O) is a set of instance and class definitions (in the sense of Definition 1) from O such that*

1. C_{st} is in $D(O)$ (the starter concept is in the result)
2. If $\mathcal{PT}$ consists of only one property directive $D_s = \langle P, n \rangle$ and $n > 0$, then
- *if C_{st} is a class in the domain of the property P, and a class $C \in O$ is in the range definition for P at C_{st}, then C is in $D(O)$ and all elements of the traversal directive result for $\langle C, \{\langle P, n-1 \rangle\} \rangle$ are in $D(O)$*
- *if C_{st} is an instance and it has a value for the property P, and this value is another class or instance $F \in O$, then F is in $D(O)$ and all elements of the traversal directive result for $\langle F, \{\langle P, n-1 \rangle\} \rangle$ are in $D(O)$.*
  If $n = \text{inf}$, then $n - 1 = \text{inf}$.

3. If $\mathcal{PT}$ consists of more than one property directive, then for each property directive $D_s \in \mathcal{PT}$, then $D(O)$ is a union of the traversal directive results for $\langle C_{st}, \{D_s\} \rangle$.

No other concepts are in D(O). ◇

This definition describes the procedure that we followed in finding the view in Example 1. Intuitively, we separate the property directives for each of the properties, and compute the traversal separately for that property directive, taking the union of the results. Note that along with the concept itself, we include its definition in the view, which includes the property values for the concept, as well as range, cardinality, and other constraint statements for the properties for which the concept serves as a domain.

Finally, we define the concept of a traversal view.

Definition 5 (Traversal View). *Given an ontology O and a Traversal View specification T, consisting of a set of traversal directives $\mathcal{TD}$, a **traversal view** $TV(O, T)$ is the union of traversal directive results for each traversal directive $D \in \mathcal{TD}$* ◇

In other words, a Traversal View is a subset of an ontology that consists of classes and instances on the path of the traversal specified in the view.

Note that if the user wants to specify a view that contains the starter concept and everything that is related to it—the transitive closure of all relations emanating from the starter concept—he needs to specify all the properties in the ontology and set the infinite depth of traversal on all of them (The user interface that we describe later makes this operation easy by having a single **"everything related"** button.)

9.2.1 Standard Properties

In addition to the domain-specific properties defined in the ontology, traversal directives can include standard relations such as `rdfs:subclassOf`, `rdf:type`, and so on. In fact, a useful directive could include a starter concept, and some specific number of levels of its subclasses (or superclasses). To include these standard ontology relations in the traversal directives, internally, we treat them as properties: Each class has a property `direct-subclasses`; the values of this property are all subclasses of the class. Similarly, there are properties `direct-superclasses`, `direct-instances`, `type-of`.[3] Therefore, these properties can be used in traversal directives just as all the regular properties.

9.2.2 View Boundary

When we extract a portion of an ontology for the view, we inevitably leave some of the definitions in the ontology incomplete. For instance, a view may include a class C_1 but not a class C_2 which is a range for a property P for C_1. Therefore, the definition of C_1 in the view is incomplete. At the same time, we may not be able to include C_2 in the view since it will violate the specific directives in the view definition.

In this case, we say that the concept C_2 in this example is in the **boundary** of the view definition: it is referenced by one of the concepts in the view but it is not itself included in the view. Maintaining the list of concepts in the view boundary is extremely useful for interactive specification of a view: the user can see which concepts are missing from the definition of the view and add them to the specification if necessary.

9.2.3 Traversal Views for Description Logics Formalisms

While we used RDF Schema to define Traversal Views, we can also specify views for description-logic ontologies, such as ontologies in OWL [3]. Classes and properties in OWL are similar to classes and properties in RDF Schema. Definitions 2, 3, 5 are trivially adapted by adjusting terminology.

In the Step 2 of Definition 4 we consider all properties that have the starter concept as part of their *domain*. There are a number of ways to restrict a property value in OWL, by using `allValuesFrom`, `someValuesFrom`, or `hasValue` restrictions. These restrictions have classes or individuals as their values. More specifically, we add the following concepts to $D(O)$ in Step 2 of the Definition:

- *If an object property P (a property that specifies relations between instances of classes) has a restriction for C_{st}, the classes specified in the restriction (whether for* `allValuesFrom`, `someValuesFrom`, `hasValue`*) are in $D(O)$*
- *If an object property P has C_{st} in its domain, the classes in the range of P are in $D(O)$*
- *If an object property P has a value for C_{st} as an individual, that value is in $D(O)$*

The view designed in such a way has all the same properties that we discuss for general Traversal Views in the next section.

[3] The Protégé ontology-editing environment, which was the platform for our implementation discussed later, takes exactly this approach.

9.3 Properties of Traversal Views

We now discuss the properties of Traversal Views that follow from Definitions 2 through 5. We consider compositional properties of single traversal directives and complete views. We then discuss completeness of our definition and computational cost of computing the view. Finally we discuss the issue of using Traversal Views to answer queries.

9.3.1 Composition of Directives and Views

Composition of traversal directives

It is easy to show, based on Definitions 2 and 4, that traversal directives are *commutative* with respect to composition: the set of concepts in the traversal view does not depend on the order of traversal directives in the view. Each directive is applied to the complete source ontology, thus traversal directive results are not affected by other directives. The result of the view is the union of the results of applying each directive.

Traversal directives are also *distributive*. Recall that a traversal directive D consists of a starter concept C_{st} and a set of property directives. Consider two traversal directives D_1 and D_2 with the same starter concept C_{st}. It follows from Definition 4 that composition of traversal directives is distributive:

$$\langle C_{st}, \mathcal{PT}_1\rangle \oplus \langle C_{st}, \mathcal{PT}_2\rangle = \langle C_{st}, \mathcal{PT}_1 \circ \mathcal{PT}_2\rangle \tag{9.1}$$

Here the composition of the sets of property directives $\mathcal{PT}_1$ and $\mathcal{PT}_2$ is the union of the directives in each set.

Composition and chaining of Traversal Views

Given an ontology O and two Traversal View specifications T_1 and T_2, we define **composition** of T_1 and T_2 as applying both T_1 and T_2 to O and then taking the union of the result:

$$T_1 \oplus T_2 = TV(O,T_1) \cup TV(O,T_2) \tag{9.2}$$

Composition defined in this way is *commutative*: the results of applying directives in one of the traversal views does not depend on the other view, since both are applied to the complete ontology O.

Chaining of Traversal Views T_1 and T_2 is the result of applying T_2 to *the result of* applying T_1 to O:

$$T_1 \circ T_2 = TV(TV(O,T_2),T_1) \tag{9.3}$$

Chaining of Traversal Views is *not commutative*: When the view T_2 applied to T_1, the set of concepts in T_2 is limited by the concepts that were included in T_1. These concepts may not include some of the concepts that would have been in T_2 if it was applied to all of the ontology O. In fact, the specification of the view T_2 may not even be a valid view specification since some of the starter concepts in traversal directives in T_2 may not even be in T_1. Thus, Traversal Views are not commutative with respect to chaining.

9.3.2 Completeness and Complexity

Completeness

Traversal Views are complete in the sense that for any subset of concepts in an ontology there is a (not necessarily unique) Traversal View that defines it. Let $\{c_1, ..., c_n\}$ be a subset of concepts. A Traversal View consisting of a set of traversal directives $\{td_1, ...td_n\}$ such that $td_k = (c_k, \emptyset)$ defines exactly the set $\{c_1, ..., c_n\}$. In other words, for each concept in the subset, we create a traversal directive containing that concept as a starter concept and no property directives.

Computational Properties

Let t be the number of traversal directives in a view definition and n be the number of concepts in the ontology. Then the running time of computing the Traversal View is $O(t*n)$. In other words, if the number of traversal directives in a view is limited by some constant c, computation of a Traversal View is linear in the size of the ontology. Indeed, computation of each traversal directive, which simply follows Definition 4, needs to examine each concept in the ontology exactly once. Suppose a concept C was examined during iteration i. If C is visited at any subsequent iterations in computing the same traversal directive, we do not need to compute the traversal directive $\langle C, \mathcal{PT}_{next}\rangle$: the depths for all the property directives will be necessarily less than at iteration i and therefore computing this directive will not add any new concepts to the view.

9.3.3 Using Traversal Views in Inference

Consider an ontology O and a Traversal View specification T. If the ontology O is consistent (for example, based on the interpretation of RDF Schema Semantics [5]), then the Traversal View $TV(O, T)$ also represents a consistent ontology based on this semantic interpretation: According to Definition 4, we do not add any new triples to $TV(O, T)$ that did not already exist in O. Further, given the fact that the constructs available in RDF Schema are monotonic, if a set of triples is consistent according to the RDF Semantics interpretation, then its subset is also consistent. Therefore, any relation between concepts in the view $TV(O, T)$ that is entailed by the view definition, is also entailed by the definitions in the full ontology O.

Note that this property of views is not true in general for Traversal Views for OWL ontologies. Consider for example the notion of disjointness in OWL. If the disjointness axiom is not included in the view, we may conclude that two entities are equivalent that should not be equivalent if disjointness is considered. We are currently working on identifying a set of syntactic constraints for an OWL ontology that would allow us to assert that any relation between two entities that is entailed by $TV(O, T)$ is also entailed by O.

9.4 Mixed-Path Traversals

In Example 1 and in the definitions in Section 9.2, we followed "homogeneous" paths to reach all nodes included in the view: All edges in a single paths were labeled with the same property.

We can also consider an alternative way of traversing the properties, which allows "mixed" paths–paths where we allow a traversal along any of the properties specified in the traversal directive. In this case, we treat the depth of a traversal for a particular property P as a specification of whether or not we should traverse the property P when we are n steps away from the starter concept, regardless of which edges we used to get to this point.

Example 3. Consider the same view specification as in the Example 1:

- $C_{st} = Lung$
- $\mathcal{PT} = \{\langle hasPart, 2\rangle, \langle containedIn, 1\rangle\}$

For the mixed-path traversal, we will generate the view in the following way:

1. `Lung` is included in the view
2. If `Lung` is in the domain of the property `hasPart` or the property `containedIn`, then all classes in the range of these properties are also included in the view
3. If `Lung` was also an instance (RDF Schema does not prevent classes from being instances as well) and had a value for properties `hasPart` and `containedIn`, those values are also included in the view

The view resulting from this process contains the traversal along the `hasPart` and `containedIn` properties of depth 1. More specifically, it will add the following classes from Figure 9.1 to the view: `LungParenchyma`, `Pulmonary-LymphaticTree`, and `ThoracicCavity`.

We then apply the same procedure to all classes in the resulting view but consider only the `hasPart` property to compute the traversal along the `hasPart` property of depth 2 (Recall that we requested a traversal of depth 1 along the `containedIn` property.) This step will add the classes `PulmonaryInterstitium` and `Mediastinum` to the view.

This view will contain all classes in Figure 9.1, including the class `Mediastinum`, which was not included in the view in Example 1, which followed only homogeneous paths. This class can be reached through the mixed path that includes both `containedIn` and `hasPart`. Intuitively, in this type of traversal, at each step n, we include everything within n nodes from the concept where we started as long as it can be reached through any combination of properties that have a traversal depth of less than n specified for them. ◇

More formally, we can define the process for computing a traversal directive result for mixed-path traversals as follows:

Definition 6 (Mixed-Path Traversal Directive Result)
Given a traversal directive $D = \langle C_{st}, \mathcal{PT} \rangle$ for an ontology O, a ***traversal directive result for mixed-path traversal*** *$D(O)$ (the result of applying directive D to O) is a set of instance and class definitions (in the sense of Definition 1) from O such that*

1. *C_{st} is in $D(O)$ (the starter concept is in the result)*
2. *For each property directive $D_s \in \mathcal{PT}$, where $D_s = \langle P, n \rangle$, $n > 0$,*
   - *if C_{st} is a class in the domain of the property P, and a class $C \in O$ is in the range definition for P at C_{st}, then C in $D(O)$*
   - *if C_{st} is an instance and it has a value for the property P, and this value is another class or instance $F \in O$, then F is in $D(O)$*
3. *$\mathcal{PT}_{next}$ is a set of property directives such that for each property directive $D_s = \langle P, n \rangle$ in $\mathcal{PT}$, the directive $\langle P, n-1 \rangle$ is in $\mathcal{PT}_{next}$. If $n = \text{inf}$, then $n - 1 = \text{inf}$. For each class or instance F that was added to $D(O)$ in step 2, the traversal directive result for a traversal directive $D_F = \langle F, \mathcal{PT}_{next} \rangle$ is in D(O).*

No other concepts are in D(O). ◇

The definition for Traversal View itself (Definition 5) remains unchanged.

We can envision use cases for both types of traversal—homogeneous and mixed-path traversals. It is easy to show that all the properties of traversal views that we discussed in Section 9.3 apply to mixed-path traversal views.

9.5 Traversal Views and Ontology Evolution

It is inevitable that ontologies change and users need to work with different versions of the same ontology. After using Traversal Views to extract a portion of an ontology, and then getting a new version of this ontology, the user should be able to determine (1) whether the view definition is still valid for the new version (that is, all the starter concepts and all the properties explicitly mentioned in the view definition are present in the new version of the ontology); and (2) whether the subset of the ontology specified by the view is unchanged.

Noy and Musen [12] have developed the PROMPTDIFF algorithm to compare different versions of the same ontology. We integrated PROMPTDIFF with Traversal Views to answer the questions above.

Given two versions of an ontology V_{old} and V_{new}, for each concept in V_{old}, PROMPTDIFF determines a corresponding concept in V_{new} if there is one. (It uses a set of heuristics to find correspondences for classes and properties that changed their names or definitions). Using this information we determine whether a view T, which was defined for V_{old}, is still valid for V_{new}. If for each concept from V_{old} that is explicitly used in T as a starter concept or a property name, there is a corresponding concept in V_{new}, then the view is valid. Furthermore, if any of the concepts explicitly used in T have changed, the user has the option of updating the view T to refer to the corresponding new concepts.

If the view has been materialized, we can use a similar technique to determine if the materialized view needs to be updated by examining the view to determine whether each concept in the view has a corresponding concept in the new version.

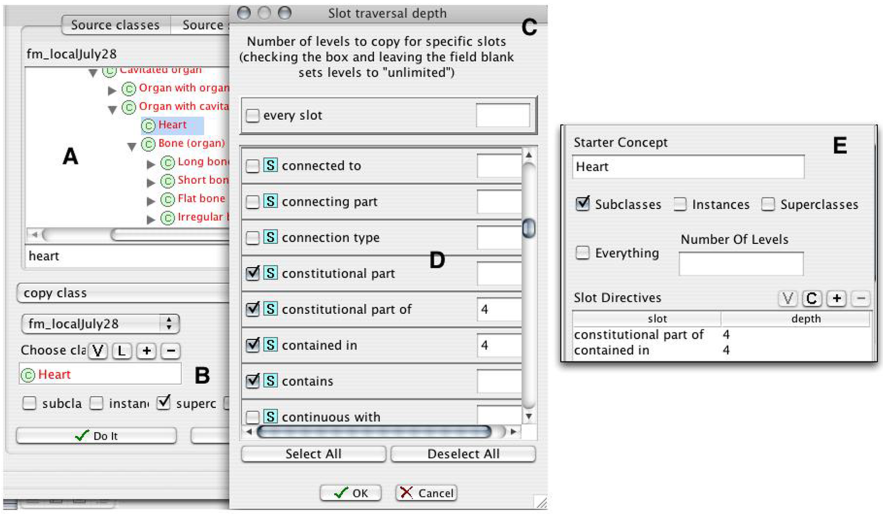
9.6 Implementation

We have implemented Traversal Views as a plugin to the Protégé ontology-development environment.[4] Protégé provides an intuitive graphical user interface for ontology development, a rich knowledge model, and an extensible architecture that provides API access both to the Protégé knowledge bases and to its user-interface components.

Figure 9.2 presents the user interface for Traversal View specification. First, the user selects a starter concept by browsing the ontology (Figure 9.2A). Then the user specifies property directives. Specification may simply include checking some of the built-in relationships, such as subclasses or superclasses (Figure 9.2B). If the user wants to specify a more detailed directive, he does this through an additional dialog (Figure 9.2C), specifying the depth of traversal for each property. In addition, the user can select the "everything" option and set the same depth of traversal for all the properties in the ontology.

After the user specifies and issues a traversal directive, we materialize the directive by copying the concepts in the view to the user's local space. In addition, we save the traversal directives as instances in an *ontology of traversal directives*.

When the user saves the results of his work, both the materialized view (the extracted portion of the source ontology) and the ontology representing the traversal



Number of levels to copy for specific slots (checking the box and leaving the field blank sets levels to "unlimited")

☐ every slot	
☐ connected to	
☐ connecting part	
☐ connection type	
☑ constitutional part	
☑ constitutional part of	4
☑ contained in	4
☑ contains	
☐ continuous with	

Starter Concept: Heart

☑ Subclasses ☐ Instances ☐ Superclasses

☐ Everything Number Of Levels:

Slot Directives

slot	depth
constitutional part of	4
contained in	4

Fig. 9.2. Specifying and examining Traversal Views. The user chooses the starter concept by browsing the hierarchy (A), specifies the traversal along the built-in properties (B), brings up a separate window (C), which lists all the properties in the ontology and built-in properties (D). The user can examine and edit the defined views through a simple form-based interface of Protégé (E).

[4] `http://protege.stanford.edu`

directive (the view definition) are saved. The user then has the option of automatically "replaying" these directives on the source ontology. Another possibility would be to "mark" concepts from the ontology that belong to the view rather than to materialize the view.

Using an ontology to store the directive is likely to lower the level of expertise required from the user to update the directive: In Protégé users browse and edit ontology instances (which is exactly what traversal directives are) through a simple form interface. To many, editing traversal directives through this interface (Figure 9.2E) is easier than writing an SQL query. Furthermore, a user can open the ontology with saved traversal directives in an ontology editor, such as Protégé, and use this simple interface to change the directives before applying them again.

9.7 Related Work

Our work is complementary to the work of using queries to define ontology views. Volz and colleagues [17, 18], for instance, define a view language based on the RQL query language [8]. In this framework, a view represents a new class or property in the ontology. The authors restrict the RQL queries that can be used in the view to ensure that the view returns only unary or binary relations (classes or properties respectively).

In theory, query-based approaches to defining ontology views are equivalent to our traversal-based approach in terms of their completeness: Just as we showed that any subset of an ontology can be produced by a combination of trivial traversal directives, we can show that any subset of an ontology can be produced by a combination of trivial queries. However, depending on what the user needs in the view and what information he can specify about the view result, a query-based or a traversal-based approach may be more appropriate.

Magkanaraki and colleagues [11] take the approach of defining query-based views further. In their RVL language, which also uses RQL for querying ontologies, the authors propose mechanisms for restructuring the original class and property hierarchies, allowing the creation of new resources, property values, classes, or properties. Therefore, a view definition includes not only the query itself, but also a set of statements that define these new structures, linking them to the query results. We can take a similar approach with Traversal Views, allowing the users to rename or reorganize the concepts in the view.

XML query languages, such as XQuery [2] are also based on traversing structures. There are, however, important differences between languages such as XQuery and Traversal Views: In XQuery, the traversal is aimed at collecting the *data* themselves rather than the schema and at presenting the data in the result according to a particular structure specified in the query. Traversal Views are aimed at traversing and collecting the schema elements themselves in the result.

Researchers in the Process Interchange Format (PIF) Project have introduced the notion of Partially Shared Views as a way of allowing different groups to extend a shared standard [9]. A Partially Specified View includes of the type hierarchy and templates, which are similar to concept definition. In this sense, Partially Specified Views are reminiscent of Traversal Views: they include definitions of concepts themselves rather that their instantiations.

The early work on traversing hypertext graphs also addressed the issue of specifying queries and views using traversal, in this case, traversal of hypergraphs. In the Gram system [1] , for instance, the authors implemented an SQL-based query language where the `FROM` clause contained a regular expression specifying a traversal. Unlike the traversals in our work, in hypertext, the results of the query where documents themselves rather than the schema. However, the traversal specification using regular expressions allowed specifying constraints on the traversal. For instance, one could traverse only the nodes that satisfied a particular set of conditions. Adding such more powerful traversal-specification mechanism for Traversal Views would be a useful extension.

A combination of views as a mechanism for extracting ontology subsets and mechanisms that allow for renaming or slight reorganization of concepts in the view is crucial to encouraging ontology reuse. This reuse, in turn, can facilitate the problem of integrating ontologies. Orbst [14] suggests creating application views of ontologies as a means of enabling applications to use standard well-developed ontologies. This approach combines views or "perspectives" with mappings between concepts in the view and in the application. The task of creating the mappings becomes much easier if the ontology to map to is smaller and contains only the concepts that are relevant to the application. Ontology views as means for extracting self-contained subsets of standard ontologies can help in this task.

9.8 Summary and Open Issues

We have presented *Traversal Views* as a way of defining an ontology view. In a Traversal View, a user specifies a subset of an ontology to include in the view by specifying starter concepts to include, the links to follow from those concepts. This mechanism enables users to extract self-contained portions of an ontology related to a particular concept or a set of concepts (such as all parts and components of the Lung).

Database researchers have done a lot of work on using views directly to *answer queries* [7]. Since database views are themselves queries, this area of research centers on reformulating the user's query to express it in terms of existing views. While Traversal Views do not map to queries directly, whether we can use their definitions directly in answering (perhaps a restricted set of) queries is an open research issue.

Traversal Views enable users to extract classes and instances to include in the view, thus simplifying the original ontology for them. However, these classes and instances can have definitions that are themselves very complex. The next step would be to allow *pruning the definitions*, hiding some of the parts or presenting some of the parts differently, based on the user's perspective.

We limited a *concept definition* for a concept C to RDF triples where C is the subject and to ranges of properties where C is a domain. This definition can be generalized to include a wider "neighborhood" of the concept C in an RDF graph. For example, it can include an *anonymous closure* of an RDF graph [15], which is computed by following graph edges, from subject to object of statements, until an anonymous RDF resource or RDF Literal is found. This extension will include, for example, all members of an RDF collection of values for a property of C in the concept definition of C. This, and similar, extensions to a concept definition, effect what a Traversal View ultimately includes.

A similar issue is the treatment of *transitive properties* for languages such as OWL. Suppose a Traversal View includes a traversal directive with a starter concept C_{st} and a property directive $\langle P, 1\rangle$, where P is a transitive property. If we issue a query for all concepts X such that $P(C_{st}, X)$, looking for all the concepts directly related to C_{st} through property P, we will get all concepts in the transitive closure of C_{st} with respect to P, rendering the depth of traversal meaningless. One possibility to handle limited traversal of transitive properties is to distinguish between explicit and inferred triples in the ontology definition.

As part of future work, we plan to perform user studies to evaluate different approaches to these issues to determine which ones provide Traversal Views that correspond most closely to users' intuition, and whether users find homogeneous or mixed-paths traversals more useful and more practical.

Finally, we would like to link query-based views, Traversal Views and other types of view-definition techniques in a unified framework to enable users to combine the different means of defining ontology views. Furthermore, we consider definition and manipulation of ontology views to be part of the general framework for ontology management. We have developed a suite of tools—PROMPT [13]—that supports various tasks in ontology management, such as comparing ontologies, merging them together, maintaining and comparing different versions, and so on. For example, the users can compare or merge views of different ontologies, find diffs between different versions of a view, integrate two views into a single ontology.

Acknowledgments

This work was conducted using the Protégé resource, which is supported by grant LM007885 from the United States National Library of Medicine. The Prompt suite is available as a plugin to the Protégé ontology-development environment at `http://protege.stanford.edu/plugins/prompt/prompt.html`

References

1. Amann, B., Scholl, M.: Gram: A graph data model and query language. In: ACM Conference on Hypertext (1992)
2. Boag, S., Chamberlin, D., Fernndez, M.F., Florescu, D., Robie, J., Simon, J.: XQuery 1.0: An XML query language. Technical report, W3C (2003)
3. Dean, M., Connolly, D., Harmelen, F.v., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., Stein, L.A.: Web ontology language (OWL) reference version 1.0 (2002), `http://www.w3.org/tr/owl-guide/`

4. Fellbaum, C. (ed.): WordNet: An Electronic Lexical Database. MIT Press, Cambridge (1998)
5. Fikes, R., McGuinness, D.L.: An axiomatic semantics for rdf, rdf-s, and daml+oil. Technical report, World Wide Web Committee (W3C) Note (2001)
6. Gruber, T.R.: A translation approach to portable ontology specification. Knowledge Acquisition 5, 199–220 (1993)
7. Halevy, A.: Answering queries using views: a survey. VLDB Journal (2001)
8. Karvounarakis, G., Alexaki, S., Christophides, V., Plexousakis, D., Scholl, M.: RQL: A declarative query language for RDF. In: Eleventh International World Wide Web Conference, Honolulu, Hawaii, USA, pp. 592–603 (2002)
9. Lee, J., Malone, T.W.: Partially shared views: a scheme for communicating among groups that use different type hierarchies. ACM Transactions on Information Systems (TOIS) 8(1), 1–26 (1990)
10. Lindberg, D., Humphreys, B., McCray, A.: The unified medical language system. Methods of Information in Medicine 32(4), 281 (1993)
11. Magkanaraki, A., Tannen, V., Christophides, V., Plexousakis, D.: Viewing the semantic web through RVL lenses. In: Fensel, D., Sycara, K.P., Mylopoulos, J. (eds.) ISWC 2003. LNCS, vol. 2870, pp. 96–112. Springer, Heidelberg (2003)
12. Noy, N.F., Musen, M.A.: PromptDiff: A fixed-point algorithm for comparing ontology versions. In: Eighteenth National Conference on Artificial Intelligence (AAAI 2002), Edmonton, Alberta (2002)
13. Noy, N.F., Musen, M.A.: The PROMPT suite: Interactive tools for ontology merging and mapping. International Journal of Human-Computer Studies 59(6), 983–1024 (2003)
14. Obrst, L., Liu, H., Wray, R.: Ontologies for corporate web applications. AI Magazine 24(3), 49–62 (2003)
15. Palmer, M., Naeve, A., Paulsson, F.: The SCAM framework: helping semantic web applications to store and access metadata. In: Bussler, C.J., Davies, J., Fensel, D., Studer, R. (eds.) ESWS 2004. LNCS, vol. 3053, pp. 167–181. Springer, Heidelberg (2004)
16. Rosse, C., Mejino, J.L.V.: A reference ontology for bioinformatics: The foundational model of anatomy. Journal of Biomedical Informatics (2004)
17. Volz, R., Oberle, D., Studer, R.: On views in the semantic web. In: 2nd International Workshop on Databases, Documents and Information Fusion (DBFUSION 2002) (2002)
18. Volz, R., Oberle, D., Studer, R.: Implementing views for light-weight web ontologies. In: IEEE Database Engineering and Application Symposium (IDEAS), Hong Kong, China (2003)
19. W3C. Resource description framework (RDF) (2000)

Part III

Connecting Existing Ontologies

Introduction to Part III

Another valid view on modularity is ontologies that differs from the one taken in part II of the book is the idea that modules are not created by splitting up a large ontology into smaller parts but by composing a number of small ontologies that have been created independently of each other into a larger model. In this scenario that was investigated in more detail in chapter 4 the original ontologies become modules in a large modular ontology. The advantage of this scenario is not easier maintenance and analysis of the overall system - in fact integrating different ontologies normally makes both more complicated compared to the individual models. The rationale for this approach is the benefit of being able to reuse knowledge that has been created by other people as well as the Data that might be associated with the ontologies to be integrated. This scenario is much closer to the original vision of the semantic web, where information sources describe their information using ontologies and information is found and reused by linking it on the level of ontologies thus creating a system of interlinked ontologies in which information can be interpreted in a uniform way.

There are several problems that have to be solved to out this vision to work: first of all, correspondences between different ontologies have to be identified manually or automatically. This process is normally referred to as ontology alignment. Once such correspondences have been found, there is a need for a way to formally represent and reason over these correspondences. As different ontologies often take contradicting views and use terms in different ways, corresponding formalisms have to be robust against problems that might arise from these differences. Finally, in order to be able to use information in such a network of ontologies across the whole system, methods have to be developed that use the formalized versions of the correspondences to retrieve information from different sources. In this part of the book, we focus on formalisms for representing correspondences between different ontologies acting as modules in such a system. The process of finding correspondences between elements in different ontologies is covered in detail in a recent book by Euzenat and Shvaiko. The retrieval of Information in a system of distributed ontologies has been discussed by different researchers, mostly in connection with Ontology-based P2P and Multi-agent system.

Being a central idea of the semantic web, the representation of links between different ontologies has already been designed into the Web Ontology Language OWL. In the definition of the language, links to other ontologies are established by using names of classes, properties or instances that are defined in other ontologies as parts of the local definitions. In addition to this, it is possible to explicitly import other ontologies for the purpose of using the definitions contained therein. It has been argued that the semantics defined for these ways of referring to other ontologies is not suitable for many applications. In particular it has been criticized that OWL takes an all or nothing attitude towards reusing definitions from other ontologies. More specifically, elements from other ontologies that are just mentioned in local definitions loose their associated meaning that might be defined in the other ontology and are just treated as new names without any further meaning. Importing an ontology, on the other hand has the same effect as copying all the definitions from the external ontologies into the local model without the possibility to leave out or alter definitions. As a result of this criticism a number of different formalisms have been proposed that try to overcome these problems by using non-standard semantics for import and links between ontologies. As it is nicely described in chapter 4 people tend to have very different ideas and intentions when importing knowledge from an external source, therefore, there cannot be a single formalism that perfectly matches every situation and different approaches have different strengthes and weaknesses.

Chapter 10 provides a comparison of different existing formalisms with the goal of understanding their differences and commonalities. The approach taken is to encode the approaches considered in a common formalism powerful enough to represent all the approaches. Distributed first-oder logic, an extension of first-order logic with a local model semantics provides such a language. The authors show that the approaches under consideration do not only make different assumptions about the nature of the relation between the information contained in different ontologies, but also that these assumptions are not compatible with each other thus supporting the claim made above that different applications need different formalisms. In the second part of the chapter, the authors describe a meta-model based approach for recording the requirements of a representation problem as a basis for selecting the appropriate formalism.

Chapter 11 presents an approach for linking ontologies that is based on the theory of ϵ-connections that have their origin in multi-dimensional logic. The underlying idea of this approach is to connect different ontologies that describe different aspects of a domain by special domain relations that establish a connection between these aspects. A typical example would be the are of medicine where one ontology could describe diseases and another one drugs. The links between classes in these two ontologies could be established by the 'treats' relation which is assigned a different meaning than an ordinary domain relation. It has been shown that this way of connecting ontologies by domain relations has theoretical advantages, in particular concerning the complexity of reasoning. The paper describes the formal foundations of the approach as well as the implementation of a specialized reasoner for linked ontologies.

Distributed Description Logics, presented in chapter 12 target a different situation in which the ontologies to be linked do not describe different aspects of a domain but the same aspects of the domain, possibly from different points of view. In this scenario, being able to deal with inconsistent definitions is of specific importance as they naturally appear when ontologies take different points of view. The solution offered by distributed description logics is based on earlier work on context-logics and allows the user to define subsumption- and equivalence relations between classes and instances in different ontologies to indicate overlap in these models which is not foreseen in the ϵ-connections model although it has been shown that theoretically distributed description logic is a special case of ϵ-connections where only one type of link exists between a pair of ontologies which is interpreted as the subsumption relation. In the chapter Serafini and Tamilin present the theory of distributed description logics and present DRAGO, a distributed reasoning system for we inferring subsumption relations across ontologies.

Distributed description logics have been criticized for their rather weak interpretation of subsumption relations that prevents some seemingly intuitive deductions across to be made. This weak semantics has been introduced to deal with the problem of inconsistent views in mind but also has some unwanted aspects such as the non-transitivity of the subsumption relation across ontologies that makes distributed description logic unsuited for modelling typical import relations between ontologies. Chapter 13 presents an approach that modifies the description logic semantics in such a way, that the typical properties of the subsumption relation such as the transitivity is retained. This is achieved by requiring a one-to-one relation between objects in the different ontologies that are linked. In this final chapter Bao and others introduce this approach called Package-Based Description logics focussing on the formal properties of the approach.

October 2008

Heiner Stuckenschmidt
Christine Parent
Stefano Spaccapietra

10

Formal and Conceptual Comparison of Ontology Mapping Languages

Saartje Brockmans[1], Peter Haase[1], Luciano Serafini[2], and Heiner Stuckenschmidt[3]

[1] Institute AIFB, University of Karlsruhe, Germany
{sbr,pha}@aifb.uni-karlsruhe.de
[2] Center for Information Technology
Fondazione Bruno Kessler
Trento, Italy
serafini@itc.it
[3] Heiner Stuckenschmidt, Computer Science Institute University of Mannheim,
A5, 6 68159 Mannheim, Germany
heiner@informatik.uni-mannheim.de

Summary. The compositional approach where several existing ontologies are connected to form a large modular ontology relies on the representation of mappings between elements in the different participating ontologies. A number of languages have been proposed for this purpose that extend existing logical languages for ontologies in a non-standard way. In this chapter, we compare different proposals for such extensions on a formal level and show that these approaches exhibit fundamental differences with respect to the assumptions underlying their semantics. In order to support application developers to select the right mapping language for a given situation, we propose a mapping metamodel that allows us to encode the formal differences on the conceptual level and facilitates the selection of an appropriate formalism on the basis of a formalism-independent specification of semantic relations between different ontologies by means of a graphical modelling language.

10.1 Motivation

The compositional approach to modular ontologies relies on appropriate definitions of interfaces between different modules to be connected. In an ideal case, these interfaces are defined at design time when modules are created in a modular fashion. In reality, we are faced with a situation where no interfaces are defined and relevant connections between ontologies have to be discovered and represented at composition time. There are two main lines of research addressing this problem. The first line is concerned with the development of methods for identifying semantic relations between elements in different ontologies. The second line of research is concerned with formalisms for encoding and using semantic relations (mappings) between ontologies. These formalisms are often based on non-standard extensions of the logics used to encode the ontologies. Examples of such mapping formalisms are [3, 7, 6, 14]. In this chapter we compare these approaches and show that they are mostly orthogonal in terms of assumptions made about the right interpretation of mapping relations. This means that the approaches cover a large variety of possible interpretations of semantic relations, but it also means that they are incompatible with each other and that the choice of a particular formalism is an important decision with significant influence on remaining options for interpreting and using mappings. Further, making the right decision with respect to a mapping formalism requires in depth knowledge of the corresponding logics and the hidden assumptions made as well as the specific needs of the application.

In order to make an informed decision about which mapping formalism to use, this decision should be made as late as possible in the modeling process because it is often not possible to decide whether a given mapping formalism is suitable for specifying all relevant connections. Therefore, mappings should first be specified on a purely informal level by just marking parts of the ontologies that are somehow semantically related. In a next step, the kind of semantic relation that exists between the elements should be specified. In order to support this process, we need a formalism-independent format for specifying mappings. On the other hand, we have to make sure that concrete mapping representations can be derived automatically from this model in order to support the implementation and use of the mappings. In order to meet these requirements, we propose a metamodel based approach to specifying ontology mappings independent from the concrete mapping formalism. In particular, we propose a Meta Object Facility-based metamodel for describing mappings between OWL DL ontologies as well as a UML profile that defines a graphical format for mapping modeling. When building the metamodel there is a natural tradeoff between coverage and precision of the metamodel: We focus on approaches that connect description logic based ontologies where mappings are specified in terms of logical axioms. This allows us to be more precise with respect to the nature and properties of mappings. At the same time, we cover a number of relevant mapping approaches that have been developed that satisfy these requirements, including the approaches mentioned in [19]. Further, the restriction to description logics allows us to use previous work on a meta-modeling approach for OWL DL [5] as a starting point.

10.1.1 Related Work and Contributions

There is some related work on meta-modeling and formalism independent modeling of mappings between conceptual models. Omelayenko introduces a model for specifying relations between heterogeneous RDF schema models for the purpose of data transformation in e-commerce [18]. The idea is to construct a separate RDF model that defines the relations in terms of so-called bridges. These bridges are accompanied by transformations that execute the translation. Maedche and others [17] describe an approach that is similar to the one of Omelayenko. They also define 'bridges' between elements of the different models and add transformation descriptions. As in the work of Omelayenko, the semantics of the bridges is only specified in terms of an RDF schema. The *mapping ontology* by Crubézy and colleagues [8] defines the structure of specific mappings and the transformation functions to transfer instances from one ontology to another. This ontology can then be used by tools to perform the transformations. The ontology provides different ways of linking concepts from the source ontology to the target ontology, transformation rules to specify how values should be changed, and conditions and effects of such rules. Our work extends and improves these approaches with respect to various aspects:

- Our approach addresses state of the art standards in the area of ontology technology, in particular OWL and rule extensions.
- Our approach is based on a sound formal foundation in terms of an encoding of different mapping formalisms in distributed first-order logic.
- We base our meta-modeling on widely used standards in the area of model-driven architectures, in particular MOF and UML.
- Our approach was designed to be able to cover most existing proposals for formal mapping approaches
- Our approach includes new insights about hidden assumptions of ontology mapping formalisms and can therefore more easily be linked to different formalisms for the sake of implementing modeled mappings.

10.1.2 Outline

We start our investigation with an informal discussion of several aspects of mapping languages including the kind of semantic relations supported, the kinds of ontology elements connected and some assumption underlying the semantics of different mapping formalisms in 10.2. In sections 10.3 and 10.4 we compare a number of mapping languages on a more formal level. For this purpose, we first introduce distributed first order logic as a unifying framework for encoding different mapping languages in section 10.3. In section 10.4, we encode different mapping language in distributed first order logic. The encoding shows that differences between mapping languages can be expressed in terms of types of axioms used to connect elements in two ontologies and basic assumptions about the relation of the domains under consideration that can be expressed in terms of a set of axioms in Distributed First Order Logic (DFOL). As these results are of purely theoretical interest so far, sections 10.5 and 10.6 are devoted to the problem of providing practical support for the selection of an appropriate mapping formalism. In particular, we propose a metamodel for ontology mappings based on an existing metamodel for OWL ontologies that captures differences and commonalities between different mapping languages on a conceptual level and can be refined to model a particular mapping language by adding constraints to the general model. Section 10.6 presents a graphical modelling language that is based on the metamodel. The language supports the representation of mappings independent of a specific formalism. We close with a discussion of the approach and topics for future work.

10.2 Ontology Mapping Formalisms

In contrast to the area of ontology languages where the Web Ontology Language OWL has become a de facto standard for representing and using ontologies, there is no agreement yet on the nature and the right formalism for defining mappings between ontologies. In a recent discussion on the nature of ontology mappings, some general aspects of mapping approaches have been identified [20]. We briefly discuss these aspects in the following and clarify our view on mappings that is reflected in the proposed metamodel with respect to these aspects.

What do mappings define ?

In this paper, we restrict our attention to declarative mapping specifications. In particular, we see mappings as axioms that define a semantic relation between elements in different ontologies. Most common are the following kinds of semantic relations:

Equivalence ($\equiv$) Equivalence states that the connected elements represent the same aspect of the real world according to some equivalence criteria. A strong form of equivalence is equality, if the connected elements represent exactly the same object.

Containment ($\sqsubseteq$, $\sqsupseteq$) Containment states that the element in one ontology represents a more specific aspect of the world than the element in the other ontology. Depending on which of the elements is more specific, the containment relation is defined in the one or in the other direction.

Overlap (o) Overlap states that the connected elements represent different aspects of the world, but have an overlap in some respect. In particular, it states that some objects described by the element in the one ontology may also be described by the connected element in the other ontology.

In some approaches, these relations are supplemented by their negative counterparts. The corresponding relations can be used to describe that two elements are *not* equivalent ($\not\equiv$), *not* contained in each other ($\not\sqsubseteq$) or *not* overlapping or disjoint respectively ($\not o$). Adding these negative versions of the relations leaves us with eight semantic relations to cover all existing proposals for mapping languages.

In addition to the type of semantic relation, an important distinction is whether the mappings are to be interpreted as extensional or as intensional relationships: In *extensional* mapping definitions, the semantic relations are interpreted as set-relations between the sets of objects represented by elements in the ontologies. In the case of *intensional* mappings, the semantic relations relate the elements directly, i.e. considering the properties of the element itself.

What are the formal properties of mappings ?

It is normally assumed that mappings preserve the 'meaning' of the two models in the sense that the semantic relation between the intended interpretations of connected elements is the one specified in the mapping. A problem with this assumption is that it is virtually impossible to verify this property. Instead, there are a number of verifiable formal properties that mappings can be required to satisfy. Examples of such formal properties are the satisfiability of the overall model, the preservation of possible inferences or the preservation of answers to queries. Often, such properties can only be stated relative to a given application context, such as a set of queries to be answered or a set of tasks to be solved. The question of what is preserved by a mapping is tightly connected to the hidden assumptions made by different mapping formalisms. A number of important assumptions that influence this aspect have been identified and formalized in [19]. The assumptions identified in the referred paper are:

- The naming of instances (are instances with the same name assumed to denote the same object?)
- The way inconsistency affects the overall system (does an inconsistency in one ontology also cause the mapped ontologies to become inconsistent)
- The assumptions about the relationships between the mapped domains (where with the *global domain assumption* both ontologies describe exactly the same set of objects, while with the *local domain assumption* the sets of objects may also be completely disjoint or overlap each other)

In [19] it has been shown that the differences between existing proposals of mapping languages for description logics can completely be described in terms of the kinds of semantic relations than can be defined and the assumptions mentioned above. This means that including these aspects in the metamodel ensures that we can model all currently existing mapping approaches and that we are able to distinguish them based on specifications that instantiate the metamodel.

Other assumptions made by approaches concerns the use of unique names for objects - this assumption is often made in the area of database integration - and the preservation of inconsistencies across mapped ontologies. In order to make an informed choice about which formalism to use, these assumptions have to be represented by the modeler and therefore need to be part of the proposed metamodel.

What do mappings connect ?

In the context of this work, we decided to focus on mappings between ontologies represented in OWL DL. This restriction makes it much easier to deal with this aspect of ontology mappings as we can refer to the corresponding metamodel for OWL DL specified in [5]. In particular, the metamodel contains the class `OntologyElement`, that represents an arbitrary part of an ontology specification. While this already covers many of the existing mapping approaches, there are a number of proposals for mapping languages that rely on the idea of view-based mappings and use semantic relations between queries to connect models, which leads to a considerably increased expressiveness.

How are mappings organized ?

The final question is how mappings are organized. They can either be part of a given model or be specified independently. In the latter case, the question is how to distinguish between mappings and other elements in the models. Mappings can be uni- or bidirectional. Further, it has to be defined whether a set of mappings is normative or whether it is possible to have different sets of mappings according to different applications, viewpoints or different matchers. In this work, we use a mapping architecture that has the greatest level of generality in the sense that other architectures can be simulated. In particular, a mapping is a set of mapping assertions that consist of a semantic relation between elements in different ontologies. Further mappings are first-class objects that exist independently of the ontologies. Mappings are directed and there can be more than one mapping between two ontologies. These choices allow considerable freedom for defining and using mappings. Approaches that see mappings as parts of an ontology can be represented by the ontology and a single mapping. If only one mapping is defined between two ontologies, this can be seen as normative, and bi-directional mappings can be described in terms of two directed mappings.

10.3 Distributed First-Order Logic

This section introduces distributed first order logic as a basis for modeling distributed knowledge bases. More details about the language including a sound and complete calculus can be found in [12].

Let $\{L_i\}_{i\in I}$ (in the following $\{L_i\}$) be a family of first order languages with equality defined over a non empty set I of indexes. Each language L_i is the language used by the i-th knowledge base (ontology). The signature of L_i is extended with a new set of symbols used to denote objects which are related with other objects in different ontologies. For each variable, and each index $j \in I$ with $j \neq i$ we have two new symbols $x^{\rightarrow j}$ and $x^{j\rightarrow}$, called *arrow variables*. Terms and formulas of L_i, also called *i-terms* and *i-formulas* and are defined in the usual way. Quantification on arrow variables is not permitted. The notation $\phi(\mathbf{x})$ is used to denote the formula ϕ and the fact that the free variables of ϕ are $\mathbf{x} = \{x_1, \ldots, x_n\}$. In order to distinguish occurrences of terms and formulas in different languages we label them with their index. The expression $\phi : i$ denotes the formula ϕ of the i-th knowledge base.

The semantics of DFOL is an extension of Local Models Semantics defined in [10]. Local models are defined in terms of first order models. To capture the fact that certain predicates are completely known by the i-th sub-system we select a sub-language of L_i containing the equality predicate, denoted as L_i^c we call the *complete fragment* of L_i. *Complete terms* and *complete formulas* are terms and formula of L_i^c and vice versa.

Definition 1 (Set of local Models). *A* set of local models *of L_i are a set of first order interpretations of L_i, on a domain $\mathbf{dom}_i$, which agree on the interpretation of L_i^c, the complete fragment of L_i.* $\diamond$

As noted in [9] there is a foundational difference between approaches that use epistemic states and approaches that use a classical model theoretic semantics. The two approaches differ as long as there is more than one model m. Using the notion of complete sublanguage L_c, however, we can force the set of local models to be either a singleton or the empty set by enforcing that $L^c = L$. Under this assumption the two ways of defining the semantics of submodels are equivalent. Using this assumption, we are therefore able to simulate both kinds of semantics in DFOL.

Two or more models can carry information about the same portion of the world. In this case we say that they *semantically overlap*. Overlapping is unrelated to the fact that the same constant appears in two languages, as from the local semantics we have that the interpretation of a constant c in L_i is independent from the interpretation of the very same constant in L_j, with $i \neq j$. Overlapping is also unrelated to the intersection between the interpretation domains of two or more contexts. Namely if $\mathbf{dom}_1 \cap \mathbf{dom}_2 \neq \emptyset$ it does not mean that L_1 and L_j overlap. Instead, DFOL explicitly represent semantic overlapping via a domain relation.

Definition 2 (Domain relation). *A* domain relation *from* $\mathbf{dom}_i$ *and* $\mathbf{dom}_j$ *is a binary relations* $r_{ij} \subseteq \mathbf{dom}_i \times \mathbf{dom}_j$. $\diamond$

Domain relation from i to j represents the capability of the j-th sub-system to represent in its domain the domain of the i-th subsystem. A pair $\langle d, d' \rangle$ being in r_{ij} means that, from the point of view of j, d in $\mathbf{dom}_i$ is the representation of d' in $\mathbf{dom}_j$. We use the functional notation $r_{ij}(d)$ to denote the set $\{d' \in \mathbf{dom}_j | \langle d, d' \rangle \in r_{ij}\}$. The domain relation r_{ij} formalizes j's subjective point of view on the relation between $\mathbf{dom}_i$ and $\mathbf{dom}_j$ and not an absolute objective point of view. Or in other words $r_{ij} \neq r_{ji}$ because of the non-symmetrical nature of mappings. Therefore $\langle d, d' \rangle \in r_{ij}$ must not be read as if d and d' were the same object in a domain shared by i and j. This facts would indeed be formalized by some observer which is external (above, meta) to both i and j. Using the notion of domain relation we can define the notion of a model for a set of local models.

Definition 3 (DFOL Model). *A DFOL* model, $\mathcal{M}$ *is a pair* $\langle \{\mathcal{M}_i\}, \{r_{ij}\} \rangle$ *where, for each* $i \neq j \in I$*:* $\mathcal{M}_i$ *is a set of local models for* L_i*, and* r_{ij} *is a domain relation from* $\mathbf{dom}_i$ *to* $\mathbf{dom}_j$. $\diamond$

We extend the classical notion of assignment (e.g., the one given for first order logic) to deal with arrow variables using domain relations. In particular, an assignment a, provides for each system i, an interpretation for all the variable, and for *some* (by not necessarily all) arrow variables as the domain relations might be such that there is no consistent way to assign arrow variables. For instance if $a_i(x) = d$ and $r_{ij}(d) = \emptyset$, then a_j cannot assign anything to $x^{i\rightarrow}$.

Definition 4 (Assignment). *Let* $\mathcal{M} = \langle \{\mathcal{M}_i\}, \{r_{ij}\} \rangle$ *be a model for* $\{L_i\}$*. An* assignment *a is a family* $\{a_i\}$ *of partial functions from the set of variables and arrow variables to* $\mathbf{dom}_i$*, such that:*

1. $a_i(x) \in \mathbf{dom}_i$*;*
2. $a_i(x^{j\rightarrow}) \in r_{ji}(a_j(x))$*;*
3. $a_j(x) \in r_{ij}(a_i(x^{\rightarrow j}))$*;*

An assignment a is admissible for a formula $\phi : i$ *if* a_i *assigns all the arrow variables occurring in* ϕ*. Furthermore, a is admissible for a set of formulas* Γ *if it is admissible* *for any $\phi : j \in \Gamma$. An assignment a is* strictly admissible *for a set of formulas Γ if it is admissible for Γ and assigns only the arrow variables that occurs in Γ.* $\diamond$

Using the notion of an admissible assignment given above, satisfiability in distributed first order logic is defined as follows:

Definition 5 (Satisfiability). *Let $\mathcal{M} = \langle \{\mathcal{M}_i\}, \{r_{ij}\} \rangle$ be a model for $\{L_i\}$, $m \in \mathcal{M}_i$, and a an assignment. An i-formula ϕ is* satisfied *by m, w.r.t, a, in symbols $m \models_D \phi[a]$ if*

1. *a is admissible for $\phi : i$ and*
2. *$m \models \phi[a_i]$, according to the definition of satisfiability for first order logic.*

$\mathcal{M} \models \Gamma[a]$ if for all $\phi : i \in \Gamma$ and $m \in \mathcal{M}_i$, $m \models_D \phi[a_i]$[1]. $\diamond$

Mappings between different knowledge bases are formalized in DFOL by a new form of constraints that involves more than one knowledge bases. These formulas that will be the basis for describing different mapping approaches are called interpretation constraints and defined as follows:

Definition 6 (Interpretation constraint). *An* interpretation constraint from $i_1, \ldots, i_n$ to *i with $i_k \neq i$ for $1 \leq k \leq n$ is an expression of the form*

$$\phi_1 : i_1, \ldots, \phi_n : i_n \rightarrow \phi : i \tag{10.1}$$

$\diamond$

The interpretation constraint (10.1) can be consider as an axiom that restrict the set of possible DFOL models to those which satisfies it. Therefore we need to define when a DFOL model satisfies an interpretation constraint.

Definition 7 (Satisfiability of interpretation constraints). *A model $\mathcal{M}$* satisfies *the interpretation constraint (10.1), in symbols $\mathcal{M} \models \phi_1 : i_1, \ldots, \phi_n : i_n \rightarrow \phi : i$ iff for any assignment a strictly admissible for $\{\phi_1 : i_1, \ldots, \phi_n : i_n\}$, if $\mathcal{M} \models \phi_k : i_k[a]$ for $1 \leq k \leq n$, then a can be extended to an assignment a' admissible for $\phi : i$ and such that $\mathcal{M} \models \phi : i[a']$.* $\diamond$

Notice that, depending on the fact that an arrow variable $x^{\rightarrow}$ occurs on the left or on the right side of the constraint, $x^{\rightarrow}$ has a universal or an existential reading. Figure 10.1 summarizes the different possible readings that will reoccur later. Notationally for any predicate P, $\|P\|_i = \bigcap_{m \in \mathcal{M}_i} m(P)$, where $m(P)$ is the interpretation of P in m.

By means of interpretation constraints on equality, we can formalize possible relations between heterogeneous domains.

[1] Since it will be clear from the context, in the rest we will use the classical satisfiability symbol $\models$ instead of $\models_D$ and we will write $m \models \phi[a]$ to mean that an i-formula ϕ is satisfied by m. In writing $m \models \phi[a]$ we always mean that of a is admissible for $\phi : i$ (in addition to the fact that m classically satisfies ϕ under the assignment a).

$a)\ \mathcal{M} \models i : P(x^{\rightarrow j}) \rightarrow j : Q(x)$ iff For all $d \in \|P\|_i$ and $\underline{\text{for all } d' \in r_{ij}(d)}$, $d' \in \|Q\|_j$

$b)\ \mathcal{M} \models i : P(x) \rightarrow j : Q(x^{i\rightarrow})$ iff For all $d \in \|P\|_i$ $\underline{\text{there is a } d' \in r_{ij}(d)}$, s.t., $d' \in \|Q\|_j$

$c)\ \mathcal{M} \models j : Q(x^{i\rightarrow}) \rightarrow i : P(x)$ iff For all $d \in \|Q\|_j$ and $\underline{\text{for all } d' \text{ with } d \in r_{ij}(d')}$, $d' \in \|P\|_i$

$d)\ \mathcal{M} \models j : Q(x) \rightarrow i : P(x^{\rightarrow j})$ iff For all $d \in \|Q\|_i$ $\underline{\text{there is a } d'}$ $\underline{\text{with } d \in r_{ij}(d')}$, s.t., $d' \in \|P\|_i$

Fig. 10.1. Implicit Quantification of Arrow Variables in Interpretation Constraints

$$\begin{aligned}
\mathsf{F}_{ij} &= \{x^{\rightarrow j} = y^{\rightarrow j} : i \rightarrow x = y : j\} \\
\mathsf{INV}_{ij} &= \left\{ \begin{array}{l} x = y^{j\rightarrow} : i \rightarrow x^{i\rightarrow} = y : j \\ x = y^{i\rightarrow} : j \rightarrow x^{j\rightarrow} = y : i \end{array} \right\} \\
\mathsf{OD}_{ij} &= \mathsf{F}_{ij} \cup \mathsf{F}_{ji} \cup \mathsf{INV}_{ij} \\
\mathsf{ED}_{ij} &= \mathsf{OD}_{ij} \cup \{x = x : i \rightarrow x^{i\rightarrow} = x^{i\rightarrow} : j\} \\
\mathsf{ID}_{ij} &= \mathsf{ED}_{ij} \cup \mathsf{ED}_{ji} \\
\mathsf{RD}_{ij} &= \left\{ \begin{array}{l} x = c : i \rightarrow x^{i\rightarrow} = c : j \\ x = c : j \rightarrow x^{j\rightarrow} = c : i \end{array} \middle| \, c \in L_i \cap L_j \right\} \\
\mathsf{IP}_{ij} &= \bot : i \rightarrow \bot : j
\end{aligned}$$

Proposition 1. *Let $\mathcal{M}$ be a DFOL model and $i \neq j \in I$.*

1. $\mathcal{M} \models \mathsf{F}_{ij}$ *iff r_{ij} is a partial function.*
2. $\mathcal{M} \models \mathsf{INV}_{ij}$ *iff r_{ij} is the inverse of r_{ji}.*
3. $\mathcal{M} \models \mathsf{OD}_{ij}$ *if $r_{ij}(= r_{ji}^{-1})$ is an isomorphism between a subset of $\mathbf{dom}_i$ and a subset of $\mathbf{dom}_j$. I.e., $\mathbf{dom}_i$ and $\mathbf{dom}_j$ (isomorphically) overlap.*
4. $\mathcal{M} \models \mathsf{ED}_{ij}$ *iff $r_{ij}(= r_{ji}^{-1})$ is an isomorphism between $\mathbf{dom}_i$ and a subset of $\mathbf{dom}_j$. I.e., $\mathbf{dom}_i$ is (isomorphically) embedded in $\mathbf{dom}_j$*
5. $\mathcal{M} \models \mathsf{ID}_{ij}$ *iff $r_{ij}(= r_{ji}^{-1})$ is an isomorphism between $\mathbf{dom}_i$ and $\mathbf{dom}_j$. I.e., $\mathbf{dom}_i$ is isomorphic to $\mathbf{dom}_j$.*
6. $\mathcal{M} \models \mathsf{RD}$, *if for every constant c of L_i and L_j, if c is interpreted in d for all $m \in \mathcal{M}_i$ then c is interpreted in $r_{ij}(d)$ for all models of $m \in \mathcal{M}_j$, and vice-versa. I.e., the constant c is rigidly interpreted by i and j in two corresponding objects.*
7. *Finally* $\mathcal{M} \models \mathsf{IP}_{ij}$ *iff $\mathcal{M}_i = \emptyset$ implies that $\mathcal{M}_j = \emptyset$. I.e., inconsistency propagates from i to j.*

10.4 Modeling Mapping Languages in DFOL

Mapping languages formalisms are based on four main parameters: local languages and local semantics used to specify the local knowledge, and mapping languages and semantics for mappings, used to specify the semantic relations between the local knowledge. In this section we focus on the second pair and as far as local languages and local semantics it is enough to notice that:

Local languages. In all approaches local knowledge is expressed by a suitable fragment of first order languages.

Local semantics. With the notable exception of [9], where authors propose an *epistemic approach* to information integration, all the other formalisms for ontology mapping assume that each local knowledge is interpreted in a (partial) state of the world and not into an epistemic state. This formally corresponds to the fact that each local knowledge base is associated with *at most one* FOL interpretation.

The first assumption is naturally captured in DFOL, by simply considering L_i to be an adequately restricted FOL language. As far as the local semantics, in DFOL models each L_i is associates with a *set of interpretations*. To simulate the single local model assumption, in DFOL it is enough to declare each L_i to be a *complete* language. This implies that all the $m \in M_i$ have to agree on the interpretation of L_i-symbols.

Notationally, $\phi, \psi, \ldots$ will be used to denote both DL expressions and FOL open formulas. If ϕ is a DL concept, $\phi(x)$ (or $\phi(x_1, \ldots, x_n)$) will denote the corresponding translation of ϕ in FOL as described in [1]. If ϕ is a role R then $\phi(x, y)$ denotes its translation $P(x, y)$, and if ϕ is a constant c, then $\phi(x)$ denote its translation $x = c$. Finally we use $\mathbf{x}$ to denote a set $x_1, \ldots, x_n$ of variables.

10.4.1 Distributed Description Logics/C-OWL

The approach presented in [2] extends DL with a local model semantics similar to the one introduced above and so-called bridge rules to define semantic relations between different T-Boxes. A distributed interpretation for DDL on a family of DL language $\{L_i\}$, is a family $\{\mathcal{I}_i\}$ of interpretations, one for each L_i plus a family $\{r_{ij}\}_{i \neq j \in I}$ of domain relations. While the original proposal only considered subsumption between concept expressions, the model was extended to a set of five semantic relations discussed below. The Semantics of the five semantic relations defined in C-OWL is the following:

Definition 8 ([4]). *Let ϕ and ψ be either concepts, or individuals, or roles of the descriptive languages L_i and L_j respectively*[2].

1. $\mathfrak{I} \models \phi : i \xrightarrow{\sqsubseteq} \psi : j$ *if* $r_{ij}(\phi^{\mathcal{I}_i}) \subseteq \psi^{\mathcal{I}_j}$;
2. $\mathfrak{I} \models \phi : i \xrightarrow{\sqsupseteq} \psi : j$ *if* $r_{ij}(\phi^{\mathcal{I}_i}) \supseteq \psi^{\mathcal{I}_j}$;
3. $\mathfrak{I} \models \phi : i \xrightarrow{\equiv} \psi : j$ *if* $r_{ij}(\phi^{\mathcal{I}_i}) = \psi^{\mathcal{I}_j}$;
4. $\mathfrak{I} \models \phi : i \xrightarrow{\perp} \psi : j$ *if* $r_{ij}(\phi^{\mathcal{I}_i}) \cap \psi^{\mathcal{I}_j} = \emptyset$;
5. $\mathfrak{I} \models \phi : i \xrightarrow{*} \psi : j$ *if* $r_{ij}(\phi^{\mathcal{I}_i}) \cap \psi^{\mathcal{I}_j} \neq \emptyset$;

[2] In this definition, to be more homogeneous, we consider the interpretations of individuals to be sets containing a single object rather than the object itself.

An interpretation for a context space is a model for it if all the bridge rules are satisfied. ◇

From the above satisfiability condition one can see that the mapping $\phi : i \xrightarrow{\equiv} \psi : j$ is equivalent to the conjunction of the mappings $\phi : i \xrightarrow{\sqsubseteq} \psi : j$ and $\phi : i \xrightarrow{\sqsupseteq} \psi : j$. The mapping $\phi : i \xrightarrow{\perp} \psi : j$ is equivalent to $\phi : i \xrightarrow{\sqsubseteq} \neg\psi : j$. And finally the mapping $\phi : i \xrightarrow{*} \psi : j$ is the negation of the mapping $\phi : i \xrightarrow{\sqsubseteq} \psi : j$. Therefore for the translation we will consider only the primitive mappings. As the underlying notion of a model is the same as for DFOL, we can directly try to translate bridge rules into interpretation constraints. In particular, there are no additional assumptions about the nature of the domains that have to be modeled. The translation is the following:

C-OWL	**DFOL**
$\phi : i \xrightarrow{\sqsubseteq} \psi : j$	$\phi(x^{\rightarrow j}) : i \rightarrow \psi(x) : j$
$\phi : i \xrightarrow{\sqsupseteq} \psi : j$	$\psi(x) : j \rightarrow \phi(x^{\rightarrow j}) : i$
$\phi : i \overset{\sqsubseteq}{\not\longrightarrow} \psi : j$	No translation

We see that a bridge rule basically corresponds to the interpretation a) and d) in Figure 10.1. The different semantic relations correspond to the usual reads of implications. Finally negative information about mappings (i.e., $\phi : i \overset{\sqsubseteq}{\not\longrightarrow} \psi : j$) is not representable by means of DFOL interpretation constraints.

10.4.2 Ontology Integration Framework (OIS)

Calvanese and colleagues in [7] propose a framework for mappings between ontologies that generalizes existing work on view-based schema integration [22] and subsumes other approaches on connecting DL models with rules. In particular, they distinguish global centric, local centric and the combined approach. Differences between these approaches are in the types of expressions connected by mappings. With respect to the semantics of mappings, they are the same and are therefore treated as one.

OIS assumes the existence of a global model g into which all local models s are mapped. On the semantic level, the domains of the local models are assumed to be embedded in a global domain. Further, in OIS constants are assumed to rigidly designate the same objects across domain. Finally, global inconsistency is assumed, in the sense that the inconsistency of a local knowledge makes the whole system inconsistent. As shown in Proposition 1, we can capture these assumptions by the set of interpretation constraints ED_{sg}, RD_{sg}, and IP_{sg}, where s is the index of any source ontology and g the index of the global ontology.

According to these assumptions mappings are described in terms of correspondences between a local and the global model. The interpretation of these correspondences are defined as follows:

Definition 9 ([7]). *Correspondences between source ontologies and global ontology are of the following three forms*

1. *$\mathcal{I}$ satisfies $\langle \phi, \psi, sound \rangle$ w.r.t. the local interpretation $\mathcal{D}$, if all the tuples satisfying ψ in $\mathcal{D}$ satisfy ϕ in $\mathcal{I}$*
2. *$\langle \phi, \psi, complete \rangle$ w.r.t. the local interpretation $\mathcal{D}$, if no tuple other than those satisfying ψ in $\mathcal{D}$ satisfies ϕ in $\mathcal{I}$,*
3. *$\langle \phi, \psi, exact \rangle$ w.r.t. the local interpretation $\mathcal{D}$, if the set of tuples that satisfies ψ in $\mathcal{D}$ is exactly the set of tuples satisfying ϕ in $\mathcal{I}$.* $\diamond$

From the above semantic conditions, $\langle \phi, \psi, exact \rangle$ is equivalent to the conjunction of $\langle \phi, \psi, sound \rangle$ and $\langle \phi, \psi, complete \rangle$. It is therefore enough to provide the translation of the first two correspondences. The definitions 1 and 2 above can directly be expressed into interpretation constraints (compare Figure 10.1) resulting in the following translation:

GLAV Correspondence	**DFOL**
$\langle \phi, \psi, sound \rangle$	$\psi(\mathbf{x}) : s \rightarrow \phi(\mathbf{x}^{s\rightarrow}) : g$
$\langle \phi, \psi, complete \rangle$	$\phi(\mathbf{x}) : g \rightarrow \psi(\mathbf{x}^{\rightarrow g}) : s$

The translation shows that there is a fundamental difference in the way mappings are interpreted in C-OWL and in OIS. While C-OWL mappings correspond to a universally quantified reading (Figure 1 a), OIS mappings have an existentially quantified readings (Figure 1 b/d). We will come back to this difference later.

10.4.3 DL for Information Integration (DLII)

A slightly different approach to the integration of different DL models is described in [6]. This approach assumes a partial overlap between the domains of the models M_i and M_j rather than a complete embedding of them in a global domain. This is captured by the interpretation constraint OD_{ij}. The other assumptions (rigid designators and global inconsistency) are the same as for OIS.

An interpretation $\mathcal{I}$ associates to each M_i a domain Δ_i. These different models are connected by inter-schema assertions. Satisfiability of interschema assertions is defined as follows:[3]

Definition 10 (Satisfiability of interschema assertions). *If $\mathcal{I}$ is an interpretation for M_i and M_j we say that $\mathcal{I}$ satisfies the interschema assertion*

[3] To simplify the definition we introduce the notation $\top^{\mathcal{I}}_{nij} = \top^{\mathcal{I}}_{ni} \cap \top^{\mathcal{I}}_{nj}$ for any $n \geq 1$. Notice that $\top^{\mathcal{I}}_{nij} = \Delta^n_i \cap \Delta^n_j$.

$\phi \sqsubseteq_{ext} \psi$, *if* $\phi^{\mathcal{I}} \subseteq \psi^{\mathcal{I}}$ $\quad$ $\phi \not\sqsubseteq_{ext} \psi$, *if* $\phi^{\mathcal{I}} \not\subseteq \psi^{\mathcal{I}}$
$\phi \equiv_{ext} \psi$, *if* $\phi^{\mathcal{I}} = \psi^{\mathcal{I}}$ $\quad$ $\phi \not\equiv_{ext} \psi$, *if* $\phi^{\mathcal{I}} \neq \psi^{\mathcal{I}}$
$\phi \sqsubseteq_{int} \psi$, *if* $\phi^{\mathcal{I}} \cap \top^{\mathcal{I}}_{nij} \subseteq \psi^{\mathcal{I}} \cap \top^{\mathcal{I}}_{nij}$
$\phi \equiv_{int} \psi$, *if* $\phi^{\mathcal{I}} \cap \top^{\mathcal{I}}_{nij} = \psi^{\mathcal{I}} \cap \top^{\mathcal{I}}_{nij}$
$\phi \not\sqsubseteq_{int} \psi$, *if* $\phi^{\mathcal{I}} \cap \top^{\mathcal{I}}_{nij} \not\subseteq \psi^{\mathcal{I}} \cap \top^{\mathcal{I}}_{nij}$
$\phi \not\equiv_{int} \psi$, *if* $\phi^{\mathcal{I}} \cap \top^{\mathcal{I}}_{nij} \neq \psi^{\mathcal{I}} \cap \top^{\mathcal{I}}_{nij}$ ◇

As before $\equiv_{est}$ and $\equiv_{int}$ are definable as conjunctions of $\sqsubseteq_{est}$ and $\sqsubseteq_{int}$, so we can ignore them for the DFOL translation. Furthermore, a distinction is made between extensional and intentional interpretation of inter-schema assertions, which leads to different translations into DFOL.

inter-schema assertions	**DFOL**
$\phi \sqsubseteq_{ext} \psi$	$\phi(\mathbf{x}) : i \rightarrow \psi(\mathbf{x}^{i\rightarrow}) : j$
$\phi \not\sqsubseteq_{ext} \psi, \phi \not\equiv_{ext} \psi$	No translation
$\phi \sqsubseteq_{int} \psi$	$\phi(\mathbf{x}^{\rightarrow j}) : i \rightarrow \psi(\mathbf{x}) : j$
$\phi \not\sqsubseteq_{int} \psi, \phi \not\equiv_{int} \psi$	No translation

While the extensional interpretation corresponds to the semantics of mappings in OIS, the intentional interpretation corresponds to the semantics of mappings in C-OWL. Thus using the distinction made in this approach we get an explanation of different conceptualizations underlying the semantics of C-OWL and OIS that use an extensional and an intentional interpretation, respectively.

10.4.4 $\mathcal{E}$-Connections

A different approach for defining relations between DL knowledge bases has emerged from the investigation of so-called $\mathcal{E}$-connections between abstract description systems [16]. Originally intended to extend the decidability of DL models by partitioning it into a set of models that use a weaker logic, the approach has recently been proposed as a framework for defining mappings between ontologies [13].

In the $\mathcal{E}$-connections framework, for every pair of ontologies ij there is a set $\mathcal{E}_{ij}$ of *links*, which represents binary relations between the domain of the i-th ontology and the domain of the j-th ontology. Links from i to j can be used to define i concepts, in a way that is analogous to how roles are used to define concepts. In the following table we report the syntax and the semantics of i-concepts definition based on links. (E denotes a link from i to j and C denotes a concept in j). The only assumption about the relation between domains is global inconsistency (see above).

In DFOL we have only one single relation between from i to j, while in $\mathcal{E}$-connection there are many possible relation. However, we can use a similar technique as used in [2] to map relations on inter-schema relations: each of the relation in $\mathcal{E}_{ij}$ acts as a r_{ij}. To represent $\mathcal{E}$-connection it is therefore enough to label each arrow variable with the proper link name. The arrow variable $x^{\xrightarrow{Own} j}$ is read as the arrow variable $x^{\rightarrow i}$ where r_{ij} is intended to be the interpretation of Own_{ij}. With this syntactic extension of DFOL concepts definitions based on links (denoted as E) can be codified in DFOL as follows:

$\mathcal{E}$-connections	DFOL
$\phi \sqsubseteq \exists E.\psi$	$\phi(x) : i \to \psi(x^{i\xrightarrow{E}}) : j$
$\phi \sqsubseteq \forall E.\psi$	$\phi(x^{\xrightarrow{E}j}) : i \to \psi(x) : j$
$\phi \sqsubseteq \geq nE.\psi$	$\bigwedge_{k=1}^{n} \phi(x_1) : i \to$ $\bigwedge_{k \neq h=1}^{n} \psi(x_k^{i\xrightarrow{E}}) \wedge x_k \neq x_h : j$
$\phi \sqsubseteq \leq nE.\psi$	$\phi(x) \wedge \bigwedge_{k=1}^{n+1} x = x_k^{\xrightarrow{E}j} : i \to$ $\bigvee_{k=1}^{n+1} \left(\psi(x_k) \supset \bigvee_{h \neq k} x_h = x_k\right) : j$

We see that like OIS, links in the $\mathcal{E}$-connections framework have an extensional interpretation. The fact, that the framework distinguishes between different types of domain relations, however, makes it different from all other approaches.

Another difference to the previous approaches is that new links can be defined on the bases of existing links similar to complex roles in DL. Syntax and semantics for link constructors is defined in the usual way: $(E^{-})^{I} = \left(E^{\mathcal{I}}\right)^{-1}$ (Inverse), $(E \sqcap F)^{I} = E^{\mathcal{I}} \cap F^{\mathcal{I}}$ (Conjunction), $(E \sqcup F)^{I} = E^{\mathcal{I}} \cup F^{\mathcal{I}}$ (Disjunction), and $(\neg E)^{I} = (\Delta_i \times \Delta_j) \setminus E^{\mathcal{I}}$ (Complement). Notice that, by means of inverse link we can define mapping of the b and d type. E.g., the e-connection statement $\phi \sqsubseteq \exists E^{-}\psi$, encodes corresponds to the DFOL bridge rules $i : \phi(x) \to j : \psi(x^{i\to})$ which is of type b). Similarly the e-connection $\phi \sqsubseteq \forall E^{-}\psi$ corresponds to a mapping of type d).

As the distinctions between different types of links is only made on the model theoretic level, it is not possible to model Boolean combinations of links. Inverse links, however, can be represented by the following axiom:

$$y = x^{\xrightarrow{E}j} : i \to y^{\xrightarrow{E^{-}}i} = x : j$$
$$y^{\xrightarrow{E^{-}}i} = x : j \to y = x^{\xrightarrow{E}j} : i$$

Finally the inclusion axioms between links, i.e., axioms of the form $E \sqsubseteq F$ where E and F are homogeneous links, i.e., links of the same $\mathcal{E}_{ij}$, can be translated in DFOL as follows:

$$x = y^{\xrightarrow{E}j} : i \to x^{i\xrightarrow{F}} = y : j$$

We can say that the $\mathcal{E}$-connections framework significantly differs from the other approaches in terms of the possibilities to define and combine mappings of different types.

10.4.5 Summary

The encoding of different mapping approaches in a common framework has two immediate advantages. The first one is the ability to reason across the different frameworks. This can be done on the basis of the DFOL translation of the different approaches using the sound and complete calculus for DFOL [11]. As there are not always complete translations, this approach does not cover all aspects of the different approaches, but as shown above, we can capture the most aspects. There are only two aspects which cannot be represented in DFOL, namely "non mappings" ($\phi : i \overset{*}{\longrightarrow} \psi : j$ in C-OWL, $\phi \not\sqsubseteq_{int} \psi$ etc. in DLII) and "complex mappings" such as complex links in $\mathcal{E}$-connection. The second benefit is the possibility to compare the expressiveness of the approaches. We have several dimensions along which the framework can differ:

Arity of mapped items[4] C-OWL allows only to align constants, concepts and roles (2-arity relations), $\mathcal{E}$-connection allows to align only 1-arity items, i.e., concepts, while DLII and OIS allow to integrate n-arity items.

Positive/negative mappings Most approaches state positive facts about mapping, e.g. that two elements are equivalent. The DLII and C-OWL frameworks also allow to state that two elements do not map ($\phi \not\equiv \psi$).

Domain relations The approaches make different assumptions about the nature of the domain. While C-OWL and $\mathcal{E}$-connections do not assume any relation between the domains, DLII assumes overlapping domains and OIS local domains that are embedded in a global domain.

Multiple mappings Only $\mathcal{E}$-connection approach supports form the definition of different types of mappings between ontologies that partition the inter-domain relations.

Local inconsistency Some approaches provide a consistent semantics also in the case in which some of the ontologies or mappings are inconsistent.

We summarize the comparison in the following table.

	Int. constr. (cf. fig. 10.1)				Mapping type			Domain	Arity	Local
	a)	b)	c)	d)	Pos.	Neg.	Mult.	relation		$\perp$
C-OWL	×			×	×	×		Het.	2	×
OIS		×		×	×			Incl.	n	
DLII	×	×			×	×		Emb.	n	
$\mathcal{E}$-Conn.	×	×	×	×	×	×	×	Het.	1	

We conclude that existing approaches make choices along a number of dimensions. These choices are obviously influenced by the intended use. Approaches intended for database integration for example will support the mapping of n-ary items that correspond to tuples in the relational model. Despite this fact, almost no work has been done on charting the landscape of choices to be made when designing a mapping approach and for adapting the approach to the requirement of the application. The work reported in this paper provides the basis for this kind of work by identifying the possible choices on a formal level. An important topic of future work is to identify possible combinations of features for mapping languages on a formal level in order to get a more complete picture of the design space of mapping languages.

[4] Due to limited space we did not discuss the encoding of mapped items in this paper.

10.5 Conceptual Comparison of Mapping Languages

As we have shown in the previous section, the differences between mapping languages can be described in terms of a fixed set of features including the kinds of semantic mappings and assumptions about the relation between the domains of interest. Other features are the kinds of language elements that can be connected by mappings. We can use these features to lift the comparison of different mapping languages from the formal to the conceptual level. For this purpose, we define a general metamodel of ontology mappings that defines structural aspects of the different formalisms and also includes attributes for defining the formal aspects that have been identified as distinguishing features of different languages.

10.5.1 A Metamodel for OWL DL Ontologies

We now review our previous work on a metamodel for OWL DL. Figure 10.2 shows the central part of the OWL DL metamodel. Among others, it shows that every element of an ontology is a subclass of the class `OntologyElement` and hence a member of an `Ontology`. The diagram of Figure 10.2 is the main part of the OWL DL metamodel but does by far not represent it fully. The metamodel is, just like OWL DL itself, a lot more extensive. Additionally, the metamodel is augmented with constraints, expressed in the Object Constraint Language ([23]), specifying invariants that have to be fulfilled by all models that instantiate the metamodel. However, for lack of space, we refer to [5] for a full specification. The metamodel for OWL DL ontologies ([5]) has a one-to-one mapping to the abstract syntax of OWL DL and thereby to its formal semantics. Our metamodel is built based on a similar approach as in [15], although the two metamodels have some fundamental differences.

Further, we have defined a metamodel for rule extensions of OWL DL. For the details, we refer the reader to [5]. In our mapping metamodel, we reuse parts of the rule metamodel, as we explain in detail in Section 10.5.2.

10.5.2 Extending the Metamodel with Mappings

We propose a formalism-independent metamodel covering OWL ontology mappings as described in Section 10.2. The metamodel is a consistent extension of our earlier work on metamodels for OWL DL ontologies and SWRL rules [5]. It has constraints defined in OCL [23] as well, which we omit here due to lack of space and instead refer to [5] for a complete reference. Figure 10.3 shows the metamodel for mappings. In the figures, darker grey classes denote classes from the metamodels of OWL DL and rule extensions. The central class in the metamodel is the class `Mapping` with four attributes. The URI, defined by the attribute `uri`, allows to uniquely identify a mapping and refer to it as a first-class object. The assumptions about the use of unique names for objects and the preservation of inconsistencies across mapped ontologies, are defined through the boolean attributes `uniqueNameAssumption` respectively `inconsistencyPreservation`. For the assumptions about the domain, we defined an attribute `DomainAssumption`. This attribute may take

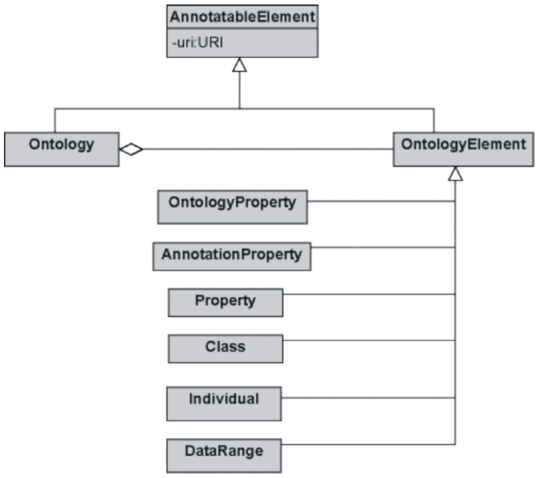


Fig. 10.2. Main Elements of the Ontology Definition Metamodel

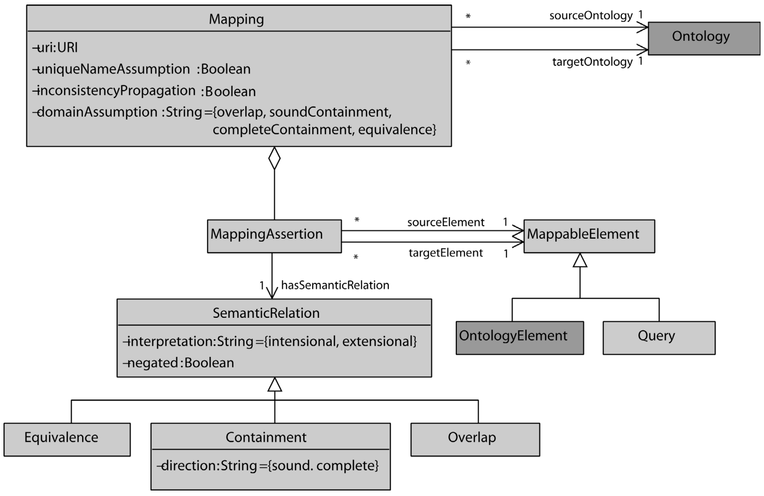

Mapping
-uri:URI
-uniqueNameAssumption :Boolean
-inconsistencyPropagation :Boolean
-domainAssumption :String={overlap, soundContainment, completeContainment, equivalence}
*
sourceOntology 1
*
targetOntology 1
Ontology
MappingAssertion
*
sourceElement 1
*
targetElement 1
MappableElement
1
hasSemanticRelation
SemanticRelation
-interpretation:String={intensional, extensional}
-negated:Boolean
OntologyElement
Query
Equivalence
Containment
-direction:String={sound. complete}
Overlap

Fig. 10.3. Metamodel for ontology mappings

specific values that describe the relationship between the connected domains: overlap, containment (in one of the two directions) or equivalence. A mapping is always defined between two ontologies. An ontology is represented by the class `Ontology` in the OWL DL metamodel. Two associations from `Mapping` to `Ontology`, `sourceOntology` and `targetOntology`, specify the source respectively the target ontology of the mapping. Cardinalities on both associations denote that to each `Mapping` instantiation, there is exactly one `Ontology` connected as source and one as target. A mapping consists of a set of mapping assertions, denoted by the MOF aggregation relationship between the two classes `Mapping` and `MappingAssertion`. The elements that are mapped in a `MappingAssertion` are defined by the class `MappableElement`. A `MappingAssertion` is defined through exactly one `SemanticRelation`, one source `MappableElement` and one target `MappableElement`. This is defined through the three associations starting from `MappingAssertion` and their cardinalities.

We defined four semantic relations along with their logical negation to be defined in the metamodel. Two of these relationship types are directly contained in the metamodel through the subclasses `Equivalence` and `Overlap` of the class `SemanticRelation`. The other two, containment in either direction, are defined through the subclass `Containment` and its additional attribute `direction`, which can be `sound` ($\sqsubseteq$) or `complete` ($\sqsupseteq$).

The negated versions of all semantic relations are specified through the boolean attribute `negated` of the class `SemanticRelation`. For example, a negated `Overlaps` relation specifies the disjointness of two elements. The other attribute of `SemanticRelation`, `interpretation`, defines whether the mapping assertion is assumed to be interpreted intensionally or extensionally. Please note that the metamodel in principle supports all semantic relations for all mappable elements, including individuals.

A mapping assertion can connect two mappable elements, which may be ontology elements or queries. To support this, `MappableElement` has two subclasses `OntologyElement` and `Query`. The former is previously defined in the OWL DL metamodel. The class `Query` reuses constructs from the SWRL metamodel. The reason for reusing large parts of the rule metamodel lies in the fact that conceptually, rules and queries are of similar nature [21]: A rule consists of a rule body (antecedent) and rule head (consequent), both of which are conjunctions of logical atoms. A query can be considered as a special kind of rule with an empty head. The distinguished variables specify the variables that are returned by the query. Informally, the answer to a query consists of all variable bindings for which the grounded rule body is logically implied by the ontology.

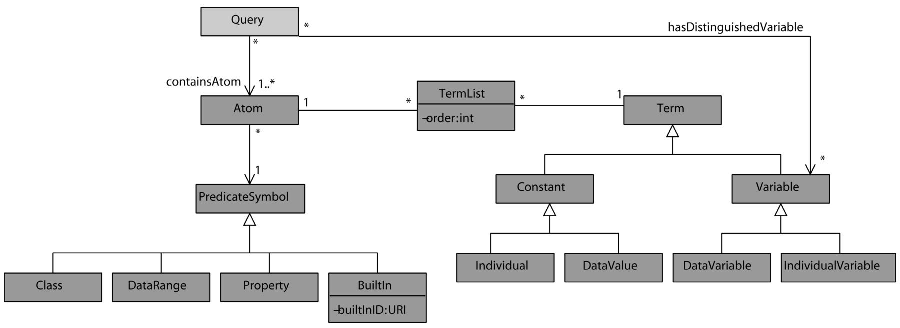


Fig. 10.4. Metamodel for ontology mappings - definition of a query

Figure 10.4 shows this connection and shows how a `Query` is composed. It depicts how atoms from the antecedent and the consequent of SWRL rules can be composed. Similarly, a `Query` also contains a `PredicateSymbol` and some, possibly just one, `Terms`. We defined the permitted predicate symbols through the subclasses `Class`, `DataRange`, `Property` and `BuiltIn`. Similarly, the four different types of terms are specified as well. The UML association class `TermList` between `Atom` and `Term` allows to identify the order of the atom terms. Distinguished variables of a query are differentiated through an association between `Query` and `Variable`.[5]

10.6 Formalism Independent Mapping Specification

The metamodel presented in the previous section allowed us to lift the comparison of mapping languages from the formal to a conceptual level and to abstract from formal details. This step does not only ease the comparison of languages it also supports the selection of an appropriate mapping language based on the actual requirements of a given application. We believe that these requirements are best captured by providing the user with the possibility of specifying semantic relations between ontologies independent of a concrete language. In this section, we present a graphical modelling language for mappings that is based on the mapping metamodel presented above. Basing the modelling language on the metamodel ensures that the resulting models can later be linked to constructs in a concrete language via the metamodel. It further allows us to test a given model against the constraints different mapping languages pose on the general metamodel and decide whether a certain language can be used to implement the graphical model.

10.6.1 A UML Profile for OWL DL Ontologies

Our UML profile is faithful to UML2 as well as to OWL DL, with a maximal reuse of features from the languages. Since the UML profile mechanism supports a restricted form of metamodeling, our proposal contains a set of extensions and constraints to UML2. This tailors UML2 such that models instantiating the OWL DL metamodel can be defined. Our UML profile has a basic mapping, from OWL class to UML class, from OWL property to binary UML association, from OWL individual to UML object, and from OWL property filler to UML object association. Extensions to UML2 consist of custom UML-stereotypes, which usually carry the name of the corresponding OWL DL language element, and dependencies.

Figure 10.5 shows a small example of an ontology using the UML profile. It contains the definition of classes `Article`, `Book` and `Thesis` as subclasses of `Publication`. The first two classes are defined to be disjoint. The ontology contains another class `Person` and its subclass `Researcher`. An association between

[5] A variable which is defined as distinguished variable in the source mappable element, must be defined as distinguished variable in the target mappable element as well.

Publication and Person denotes the object property authorOf, from which domain and range are defined via an association class. Furthermore, the ontology defines object properties between Publication and Topic, and between Topic and Name. Finally, the ontology contains some instances of its classes and object property. For a discussion of all details of the UML profile for OWL DL ontologies, we refer to [5].

Another small ontology of the same domain is presented in Figure 10.6. In typical use cases such as data translation, data integration, etc. mappings between the two ontologies would have to be defined, as described in the earlier sections. The following sections present a metamodel and UML profile for the definition of these ontology mappings.

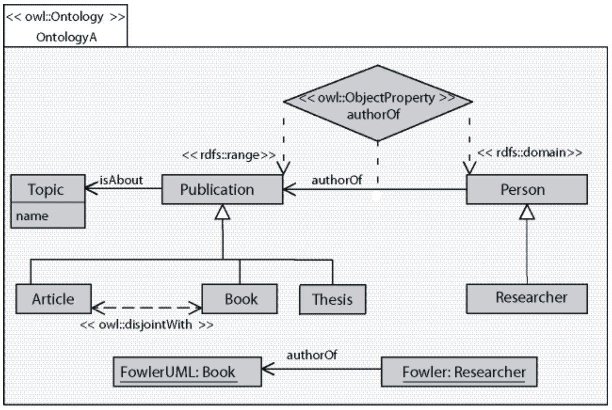


Fig. 10.5. A First Sample Ontology Depicted using the UML Profile for the Ontology Metamodel

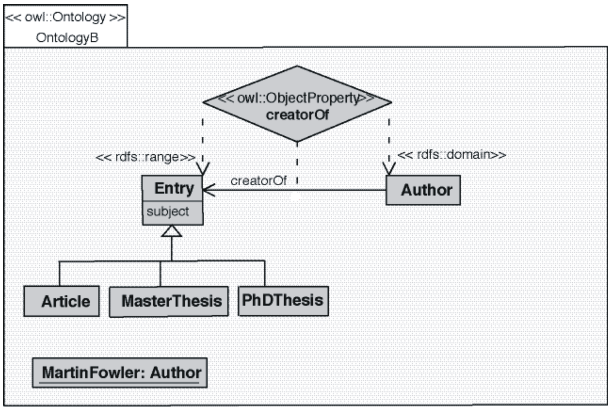


Fig. 10.6. A Second Sample Ontology Depicted using the UML Profile for the Ontology Metamodel

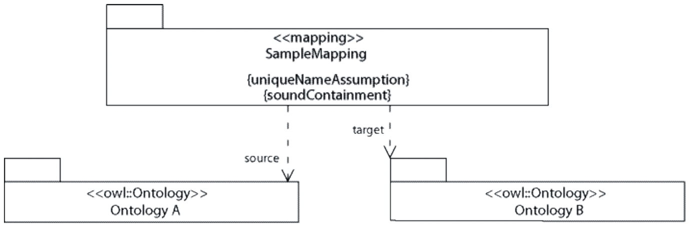


Fig. 10.7. Visual notation for a mapping between two ontologies

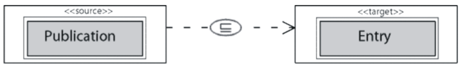


Fig. 10.8. Sample containment relation between two concepts

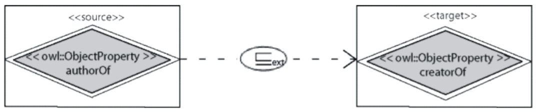


Fig. 10.9. Sample extensional containment relation between two properties

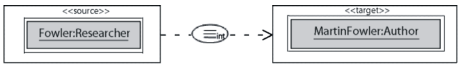


Fig. 10.10. Sample intensional equivalence relation between two individuals

10.6.2 A UML Profile for Ontology Mappings

This section describes the UML profile as a visual notation for specifying ontology mappings, based on the metamodel discussed in Section 10.5.2. The UML profile is consistent with the design considerations taken for the previously defined UML profiles for OWL ontologies and rule extensions.

First of all, users specify two ontologies between which they want to define mappings. The visual notation for this as defined in our profile, is presented in Figure 10.7. Just as for ontologies as collections of ontology elements, we apply the UML grouping construct of a package to represent mappings as collections of mapping assertions. Attributes of the mapping, like the domain assumption, are represented between curly brackets. In Figure 10.8, a source concept `Publication` is defined to be more specific than the target concept `Entry`. The example in Figure 10.9 relates two properties `authorOf` and `creatorOf` using an extensional containment relationship. Figure 10.10 models `Researcher Fowler` and `Author MartinFowler` as two equivalent instances. Both source and target elements of mapping assertions are represented in a box, connected to each other via a dependency with the corresponding symbol of the semantic relation. In the first step of the process, when users just mark elements being semantically related without specifying the type of semantic relation, the dependency does not carry any relation symbol. Stereotypes in the two boxes denote source- and target ontology. Like defined in the metamodel, these mapped elements can be any element of an ontology (metaclass `OntologyElement`) or a query (metaclass `Query`). They are represented like defined in the UML profile for OWL and rules. The parts of the mappable elements which are effectively being mapped to each other, are denoted via a double-lined box, which becomes relevant if the mapped elements are more complex constructs, as explained in the following.

A more complex example mapping assertion is pictured in Figure 10.11. The example defines that the union of the classes `PhDThesis` and `MasterThesis`, is equivalent to the class `Thesis`.

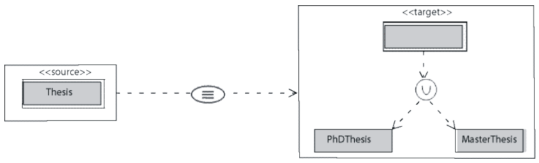


Fig. 10.11. Sample equivalence relation between complex class descriptions

Figure 10.12 shows another example of an equivalence relation between two expressions. It specifies that the class which is connected to the class `Publication` via a property `authorOf` with the `someValuesFrom` restriction, is equivalent to the class `Author`.

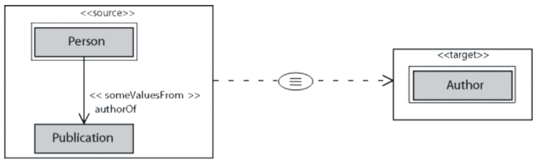


Fig. 10.12. Sample equivalence relation between complex class descriptions

Figure 10.13 shows an example of an equivalence relation between two queries. The first query is about a Publication X with a Topic Y named Z. The target query is about an Entry X with subject Z. The mapping assertion defines the two queries to be equivalent. The effective correspondences are established between the two distinguished variables X and Z, again denoted with a double-lined box.

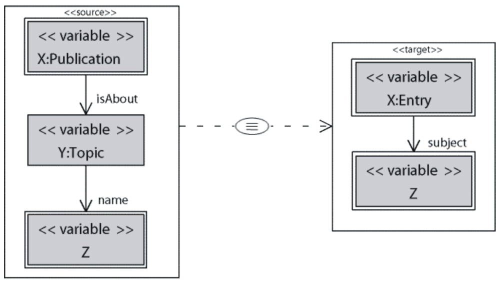


Fig. 10.13. Sample equivalence relation between two queries

10.7 Discussion and Conclusions

In this chapter, we have presented a comprehensive comparison of ontology mapping languages on a formal and a conceptual level. Based on an encoding in distributed first order logic, we have shown that existing mapping languages differ in a number of quite fundamental assumptions that make them largely incompatible with each other. We have concluded that the choice of a suitable mapping formalism is a critical success factor for a successful composition of ontologies. It is clear that this choice should be based on the formal characteristics of the languages. In order to support the choice of a language based on these characteristics we have presented a metamodel and a graphical modelling language to support formalism independent graphical modeling of mappings between OWL ontologies and their required characteristics. The metamodel ties in with previous work on similar metamodels for OWL DL and rule extensions and the results of the formal analysis of mapping languages. In order to be able to provide support not only for the acquisition of mappings but also for their implementation in one of the existing formalisms, three additional steps have to be taken. In a first step, we have to link the abstract metamodel to concrete mapping formalisms. This can best be done by creating specializations of the generic metamodel that correspond to individual mapping formalisms. This normally means that restrictions are added to the metamodel in terms of OCL constraints that formalize the specific properties of the respective formalism. In a second step, we have to develop a method for checking the compatibility of a given graphical model with a particular specialization of the metamodel. This is necessary for being able to determine whether a given model can be implemented with a particular formalism. Provided that specializations are entirely described using OCL constraints, this can be done using an OCL model checker. Finally, we have to develop methods for translating a given graphical model into an appropriate mapping formalism. This task can be seen as a special case of code generation where instead of executable code, we generate a formal mapping model that can be operationalized using a suitable inference engine.

References

1. Borgida, A.: On the relative expressiveness of description logics and predicate logics. Artificial Intelligence 82, 353–367 (1996) (research note)
2. Borgida, A., Serafini, L.: Distributed description logics: Assimilating information from peer sources. Journal of Data Semantics 1, 153–184 (2003)
3. Bouquet, P., Giunchiglia, F., van Harmelen, F., Serafini, L., Stuckenschmidt, H.: C-OWL: Contextualizing ontologies. In: Fensel, D., Sycara, K.P., Mylopoulos, J. (eds.) ISWC 2003. LNCS, vol. 2870, pp. 164–179. Springer, Heidelberg (2003)
4. Bouquet, P., Giunchiglia, F., van Harmelen, F., Serafini, L., Stuckenschmidt, H.: Contextualizing ontologies. Journal on Web Semantics 1(4), 325–343 (2004)
5. Brockmans, S., Haase, P.: A Metamodel and UML Profile for Networked Ontologies – A Complete Reference. Technical report, Universität Karlsruhe (April 2006), http://www.aifb.uni-karlsruhe.de/WBS/sbr/publications/ontology-metamodeling.pdf
6. Calvanese, D., Giacomo, G.D., Lenzerini, M.: Description logics for information integration. In: Kakas, A.C., Sadri, F. (eds.) Computational Logic: Logic Programming and Beyond. LNCS, vol. 2408, pp. 41–60. Springer, Heidelberg (2002)
7. Calvanese, D., Giacomo, G.D., Lenzerini, M.: A framework for ontology integration. In: Cruz, I., Decker, S., Euzenat, J., McGuinness, D. (eds.) The Emerging Semantic Web, pp. 201–214. IOS Press, Amsterdam (2002)
8. Crubézy, M., Musen, M.A.: Ontologies in support of problem solving. In: Staab, S., Studer, R. (eds.) Handbook on Ontologies, pp. 321–342. Springer, Heidelberg (2003)
9. Franconi, E., Tessaris, S.: Rules and queries with ontologies: a unified logical framework. In: Ohlbach, H.J., Schaffert, S. (eds.) PPSWR 2004. LNCS, vol. 3208, pp. 50–60. Springer, Heidelberg (2004)
10. Ghidini, C., Giunchiglia, F.: Local model semantics, or contextual reasoning = locality + compatibility. Artificial Intelligence 127(2), 221–259 (2001)
11. Ghidini, C., Serafini, L.: Distributed first order logics. In: Gabbay, D.M., De Rijke, M. (eds.) Frontiers of Combining Systems 2, pp. 121–139. Research Studies Press Ltd. (2000)
12. Ghidini, C., Serafini, L.: Distributed first order logic - revised semantics. Technical report, ITC-irst (January 2005)
13. Grau, B.C., Parsia, B., Sirin, E.: Working with multiple ontologies on the semantic web. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 620–634. Springer, Heidelberg (2004)
14. Haase, P., Motik, B.: A mapping system for the integration of OWL-DL ontologies. In: Proceedings of the ACM-Workshop: Interoperability of Heterogeneous Information Systems (IHIS 2005) (November 2005)
15. IBM, Sandpiper Software. Ontology Definition Metamodel, Fourth Revised Submission to OMG (November 2005)
16. Kutz, O., Lutz, C., Wolter, F., Zakharyaschev, M.: E-connections of abstract description systems. Artificial Intelligence 156(1), 1–73 (2004)
17. Maedche, A., Motik, B., Silva, N., Volz, R.: MAFRA - a mapping framework for distributed ontologies. In: Gómez-Pérez, A., Benjamins, V.R. (eds.) EKAW 2002. LNCS, vol. 2473, p. 235. Springer, Heidelberg (2002)
18. Omelayenko, B.: RDFT: A mapping meta-ontology for business integration. In: Proceedings of the Workshop on Knowledge Transformation for the Semantic Web (KTSW 2002) at the 15-th European Conference on Artificial Intelligence, Lyon, France, pp. 76–83 (2002)

19. Serafini, L., Stuckenschmidt, H., Wache, H.: A formal investigation of mapping languages for terminological knowledge. In: Proceedings of the 19th International Joint Conference on Artificial Intelligence - IJCAI 2005, Edinburgh, UK (August 2005)
20. Stuckenschmidt, H., Uschold, M.: Representation of semantic mappings. In: Kalfoglou, Y., Schorlemmer, M., Sheth, A., Staab, S., Uschold, M. (eds.) Semantic Interoperability and Integration. Dagstuhl Seminar Proceedings, Germany, vol. 04391. IBFI, Schloss Dagstuhl (2005)
21. Tessaris, S., Franconi, E.: Rules and queries with ontologies: a unifying logical framework. In: Horrocks, I., Sattler, U., Wolter, F. (eds.) Description Logics. CEUR Workshop Proceedings, vol. 147, CEUR-WS.org. (2005)
22. Ullman, J.D.: Information integration using logical views. In: Afrati, F.N., Kolaitis, P.G. (eds.) ICDT 1997. LNCS, vol. 1186, pp. 19–40. Springer, Heidelberg (1997)
23. Warmer, J., Kleppe, A.: Object Constraint Language 2.0. MITP Verlag (2004)

11

Ontology Integration Using $\mathcal{E}$-Connections

Bernardo Cuenca Grau[1], Bijan Parsia[2,3], and Evren Sirin[2]

[1] University of Oxford, UK
berg@comlab.ox.ac.uk
[2] University of Manchester, UK
bparsia@cs.man.ac.uk
[3] Clark & Parsia, USA
{bijan,evren}@clarkparsia.com

Summary. The standardization of the Web Ontology Language, OWL, leaves (at least) two important issues for Web-based ontologies unsatisfactorily resolved, namely how to represent and reason with multiple distinct, but linked ontologies, and how to enable effective knowledge reuse and sharing on the Semantic Web. In this paper, we present a solution for these problems based on $\mathcal{E}$-Connections. We aim to use $\mathcal{E}$-Connections to provide modelers with suitable means for developing Web ontologies in a modular way and to provide an alternative to the *owl:imports* construct.

With such motivation, we present in this paper a syntactic and semantic extension of the Web Ontology language that covers $\mathcal{E}$-Connections of OWL-DL ontologies. We show how to use such an extension as an alternative to the *owl:imports* construct in many modeling situations. We investigate different combinations of description logics for which it is possible to design and implement reasoning algorithms, well-suited for optimization. Finally, we provide support for $\mathcal{E}$-Connections in both an ontology editor, Swoop, and an OWL reasoner, Pellet.

11.1 Motivation

The Semantic Web architecture has been envisioned as a set of new languages that are being standardized by the World Wide Web Consortium (W3C). Among these languages, the Web Ontology Language (OWL) plays a prominent role, and Description Logics have deeply influenced its design and standardization [1][20]. Two of the three variants, or dialects, of OWL, namely OWL-Lite and OWL-DL, correspond to the logics $\mathcal{SHIF}(\mathcal{D})$ and $\mathcal{SHOIN}(\mathcal{D})$, respectively [20] [26] [28].

The acceptance of OWL as a Web standard will yield to the rapid proliferation of DL ontologies on the Web and it is envisioned that, in the near future, the Semantic Web will contain a large number of independently developed ontologies. The standardization of OWL, however, leaves two important issues for Web-based ontologies unsatisfactorily resolved, namely how to represent and reason with multiple distinct, but linked ontologies, and how to enable effective knowledge reuse and sharing on the Semantic Web.

First, in order to provide support for integrating ontologies, OWL defines the *owl:imports* construct, which allows to include by reference in an ontology the axioms contained in another ontology, published somewhere on the Web and identified by a global name (a URI). The functionality provided by this construct, however, is unsatisfactory for a number of reasons [8]:

- The only way that the *owl:imports* construct provides for using concepts from a different ontology is to bring into the original ontology *all* the axioms of the imported one. Therefore, the only difference between *copying and pasting* the imported ontology into the importing one and using an *owl:imports* statement is the fact that with imports both ontologies stay in different files. This certainly provides some *syntactic* modularity , but not a *logical* modularity, which would be indeed more desirable.
- The entities of an ontology, such as classes and properties, are, as the ontologies themselves, identified by unique names (URIs) on the Semantic Web. For example, suppose that we are developing an ontology about "People" and we want to define the concept of a "Dog Owner". It may seem natural for such a purpose to use the URI of a certain class "Dog", that appears in an ontology about "Pets" that we have found on the Web. We may think then that we are committing to the meaning of "Dog" in that ontology (a dog is an animal, for example). Nevertheless, if we use the URI for "Dog" without importing the corresponding ontology, we are bringing *nothing* from the meaning of the term in the the foreign ontology, while if we import it, we are bringing all the axioms of the "Pet" ontology to our logical space, even if we are only interested in dogs, and not in cats or hamsters.
- The use of *owl:imports* results in a flat ontology, i.e., none of the imported axioms or facts retain their context. While it is possible to track down the originator(s) of some assertions by inspecting the imported ontology, OWL reasoning does not take such context into account.

Second, enabling knowledge reuse and sharing has always been a major goal of the Web Ontology Working Group. Ontology engineering is a time-consuming task. As more ontologies are built and become available, and as the size of ontologies grows, knowledge sharing and reuse become important issues. On the one hand, when ontologies grow, they become harder for the reasoners to process and for humans to understand, and also harder to reuse. On the other hand, as more ontologies become available, the advantages of reusing existing ontologies become more apparent. In order to make reuse and sharing easier, ontologies should be designed as mostly independent and self-contained modules [30][29]. Intuitively, a module should contain information about a self-contained application domain. Then, suitable means should be provided for integrating and connecting those modular ontologies.

In this paper, we present an approach for tackling these two issues based on $\mathcal{E}$-Connections. The $\mathcal{E}$-Connections technique [25] [24] is a method for combining logical languages. The main motivation of $\mathcal{E}$-Connections is to combine decidable logics in such a way that the resulting combined formalism remains decidable, although the increase of expressivity may result in a higher worst-case complexity.

In this paper, we present an extension of the Web Ontology language that covers $\mathcal{E}$-Connections of OWL-DL ontologies. We show how such an extension can be used to achieve modular ontology development on the Semantic Web and how $\mathcal{E}$-Connections provide a suitable framework for integration of Web ontologies. We investigate the use of $\mathcal{E}$-Connections as an alternative to the *owl:imports* construct in many modeling situations. We show that, in the case of $\mathcal{E}$-Connections of OWL-DL ontologies, it is possible to design reasoning algorithms, well-suited for implementation and optimization. We prove that these algorithms can be implemented as an extension of current reasoners and, contrary to what is thought about $\mathcal{E}$-Connections, we argue that they have a potential for enhancing performance, since they suggest new optimization techniques. Finally, we provide support for $\mathcal{E}$-Connections in both an OWL ontology editor (Swoop [23]) and an OWL reasoner (Pellet [27]).

11.2 $\mathcal{E}$-Connections of Web Ontologies

An $\mathcal{E}$-Connection is a knowledge representation language defined as a combination of other logical formalisms. Each of the component logics has to be expressible in the Abstract Description System (ADS) framework [2], which includes Description Logics (and hence OWL-DL), some temporal and spatial logics, Modal and Epistemic logics. Obviously, different component logics will give rise to different combined languages, with different expressivity and computational properties.

$\mathcal{E}$-Connections were originally introduced in [31] as a way to increase the expressivity of each of the component logics, while preserving the decidability of the reasoning services. Thus, $\mathcal{E}$-Connections were conceived for providing a trade-off between the expressivity gained and the computational robustness of the combination. Here, we will use $\mathcal{E}$-Connections as a language for defining and instantiating combinations of OWL-DL ontologies. We will restrict ourselves to OWL-DL, since OWL-Full is beyond the Abstract Description System framework. From now on in the paper, whenever we mention *OWL*, we will implicitly refer to OWL-DL. In this section, we assume that the reader is familiar with OWL.

An $\mathcal{E}$-Connection is a set of "connected" ontologies. An $\mathcal{E}$-Connected ontology[1] contains information about classes , properties and their instances , but also about a new kind of properties, called *link properties* , which establish the connection between the ontologies.

Link properties are similar in spirit to datatype properties in the sense that they are used for combining information from different domains (the actual application domain of the ontology and the domain of datatypes , in the case of datatype properties). The coupling between datatypes and the ontology is always achieved through

[1] In this paper, we use "$\mathcal{E}$-Connection Language" to denote a formalism, i.e. a logic; we will use "$\mathcal{E}$-Connection" to denote a knowledge base written in such a language. These knowledge bases are composed of a set of "connected" ontologies, for which we will use the term "$\mathcal{E}$-Connected ontology" or "component ontology".

restrictions on datatype properties. For example, a “retired person” can be defined in OWL as a person whose age is greater than 65, by using a class (“Person”) in the ontology and a restriction on a datatype property “age” with value “greater than 65”. Both from a logical and from a modeling perspective, the domain of the ontology and the domain of datatypes are separate: from a modeling perspective, the (application) domain of “persons” does not overlap with the (application) domain of “numbers”; from a logical perspective, in OWL, the domain where classes, properties and individuals in the ontology are interpreted is disjoint from the domain of datatypes, and datatype properties are interpreted as binary relations with the first element belonging to the domain of the ontology and the second on the domain of the datatypes. In the same vein, link properties allow to create classes in a certain ontology based on information from a *different* ontology. For example, a *GraduateStudent* in an ontology about “people” could be defined as a student who is enrolled in at least one graduate course, by using the class *Student* in the people ontology and a *someValuesFrom* restriction on the link property *enrolledIn* with value *GraduateCourse*, which would be a class in a different ontology dealing with the domain of “academic courses”.

Link properties are logically interpreted as binary relations, where the first element belongs to its “source” ontology and the second to its “target ontology”. Conceptually, a link property will be defined and used in its “source” ontology. For example, the link property “enrolledIn” would be defined as a link property in the “people” ontology with target ontology “academic courses”.

An $\mathcal{E}$-Connected ontology can be roughly described as an OWL-DL ontology, extended with the ability to define link properties and construct new classes in terms of restrictions on them. An $\mathcal{E}$-Connection is then defined as a set of $\mathcal{E}$-Connected ontologies. From the modeling perspective, each of the component ontologies in an $\mathcal{E}$-Connection is modeling a different application domain, while the $\mathcal{E}$-Connection itself models the *union* of all these domains. For example, an $\mathcal{E}$-Connection could be used to model all the relevant information referred to a certain university, and each of its component ontologies could model, respectively, the domain of people involved in the university, the domain of schools and departments, the domain of courses, etc.

11.2.1 Basic Elements

In order to illustrate the basic elements of an $\mathcal{E}$-Connection, let us consider the following application domains, that we want to formalize: let D_1 be the domain of “travel accommodations”, D_2 the domain of “leisure activities”, D_3 the domain of “travel destinations”, and D_4 the domain of “people”. We want to use an $\mathcal{E}$-Connection to model the union of these domains, i.e., the domain of “tourism”.

We want to model each application domain in a different component of the $\mathcal{E}$-Connection and then use link properties to talk about their relationships.

As in OWL, each $\mathcal{E}$-Connected ontology is written in a different file, and the vocabularies being used in each of them can be specified by a set of namespace declarations and entity definitions. In this Section, we use the namespace *ec:* to denote the new vocabulary we introduce for $\mathcal{E}$-Connections.

For our domains, we create the following root classes[2]:

```
(accommodations)
  <owl:Class rdf:ID= "Accommodation"/>

(activities)
<owl:Class rdf:ID= "Activity"/>

(destinations)
<owl:Class rdf:ID= "Destination"/>

(people)
<owl:Class rdf:ID= "Person"/>
```

We would like to define classes like *BudgetDestination* (a travel destination which provides a choice of budget accommodations), a *CaribbeanHotel* (a hotel accommodation offered at a Caribbean destination), and a *SportsDestination* (a destination that offers a variety of activities related to sport).

In order to attain this goal, we define a set of *link properties*, i.e. properties that relate elements of the different domains. For example, the links *providesAccommodation* and *offersActivity* relate the domain of "destinations" to the domain of "accommodations" and "activities" respectively.

```
(destinations)
<ec:LinkProperty rdf:ID="providesAccommodation">
  <ec:foreignOntology rdf:resource="&acco;"/>
  <rdfs:domain rdf:resource="#Destination"/>
  <rdfs:range>
    <ec:ForeignClass rdf:about="&acco;#Accommodation">
      <ec:foreignOntology rdf:resource="&acco;"/>
    </ec:ForeignClass>
  </rdfs:range>
</ec:LinkProperty>
<ec:LinkProperty rdf:ID="offersActivity">
  <ec:foreignOntology rdf:resource="&activities;"/>
  <rdfs:domain rdf:resource="#Destination"/>
  <rdfs:range>
    <ec:ForeignClass rdf:about="&activities;#Activity">
      <ec:foreignOntology rdf:resource="&activities;"/>
    </ec:ForeignClass>
  </rdfs:range>
</ec:LinkProperty>
```

A link property is a binary relation between instances of classes, which belong to different $\mathcal{E}$-Connected ontologies. The *source* of a link property is the ontology in which it has been declared; the *target* of the link is the ontology specified in the *owl:foreignOntology* tag in the declaration.

The first element of the relation always belongs to an instance of a class in the source ontology. In the example, both *providesAccommodation* and *offersActivity* have been defined in the "destinations" ontology. The second element of the relation corresponds to an individual in the target ontology, i.e. the "accommodations"

[2] In brackets, we specify the ontology each class has been defined in; we use this informal notation along this section for clarity and brevity, in order to avoid including the namespace and ontology headers of each ontology in the combination.

ontology in the case of *providesAccommodation* and the *activities* ontology in the case of *offersActivity*.

The definition of a link property *must* include a *single owl:foreignOntology* tag. As in the case of object properties, link properties can be assigned a domain and a range. For example, the link property *offersActivity* relates instances of the class *Destination* to instances of the class *Activity*. The class specified as a range of a link property *must* be declared as a class in the target ontology. In the source ontology, such a class can be declared as "foreign" using the *owl:ForeignClass* tag.

A URI cannot be used in a given ontology both as "local" (declared as a class in the ontology using the owl:Class tag) and "foreign"; if this happens, a reasoner *must* treat such an ontology as *inconsistent*.

A link property can be defined as functional or inverse functional, with the usual meaning.

However, as opposed to object properties in OWL, a link property cannot be tagged as transitive or symmetric. Note that, within an $\mathcal{E}$-Connection, a link property is defined in a certain "source" ontology and points to a specific "target ontology". In other words, each link property connects (only) two ontologies in a given $\mathcal{E}$-Connection.

Restrictions on link properties can be used to generate new concepts. For example, we can define a "budget destination" as a travel destination that offers at least one kind of budget accommodation:

```
(destinations)
 <owl:Class rdf:ID="BudgetDestination">
   <owl:intersectionOf rdf:parseType="Collection">
      <owl:Class rdf:about="#Destination"/>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#providesAccommodation"/>
        <owl:someValuesFrom>
           <ec:ForeignClass rdf:about="&acco;BudgetAccommodation">
              <ec:foreignOntology rdf:resource="&acco;"/>
           </ec:ForeignClass>
        </owl:someValuesFrom>
      </owl:Restriction>
   </owl:intersectionOf>
 </owl:Class>
```

Similarly, we can define a *CinemaLover* as a person who likes Cinema:

```
(persons)
<owl:Class rdf:ID="CinemaLover">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Person"/>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#likesActivity"/>
      <owl:someValuesFrom>
        <ec:ForeignClass rdf:about="&activities;CinemaActivity">
          <ec:foreignOntology rdf:resource="&activities;"/>
        </ec:ForeignClass>
      </owl:someValuesFrom>
    </owl:Restriction>
  </rdfs:subclassOf>
</owl:Class>
```

Using an “allValuesFrom” restriction we can define a *FanaticCinemaLover* as a *CinemaLover* who likes no activity other than cinema:

```
(persons)
<owl:Class rdf:ID="FanaticCinemaLover">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#CinemaLover"/>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#likesActivity"/>
      <owl:allValuesFrom>
        <ec:ForeignClass rdf:about="&activities;CinemaActivity">
          <ec:foreignOntology rdf:resource="&activities;"/>
        </ec:ForeignClass>
      </owl:allValuesFrom>
    </owl:Restriction>
  </rdfs:subclassOf>
</owl:Class>
```

where in the “activities” ontology we would define the class *CinemaActivity* as a subclass of *Activity*:

```
(activities)
<owl:Class rdf:ID="CinemaActivity">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Activity"/>
  </rdfs:subClassOf>
</owl:Class>
```

Cardinality restrictions on link properties allow to constrain the number of objects linked by the connecting relations. For instance, we can define a *SportsDestination* as a travel destination that offers more than 10 different sports activities:

```
(destinations)
<owl:Class rdf:ID = "SportsDestination">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Destination"/>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="offersSportActivity"/>
      <owl:minCardinality rdf:datatype="&xsd;Integer">10
      </owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

The “hasValue” restriction on link properties allows to specify classes in an $\mathcal{E}$-Connected ontology based on the existence of a particular individual in a different ontology. For example, we can define the class *SurfingDestination* as a travel destination that offers surfing as one of their activities, where *surfing* is an instance of the class *SportsActivity*.

```
(destinations)
<owl:Class rdf:ID = "SurfingDestination">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Destination"/>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource = "offersSportActivity"/>
```

```
          <owl:hasValue>
            <ec:ForeignIndividual rdf:about="&activities;surfing">
              <ec:foreignOntology rdf:resource="&activities;"/>
            </ec:ForeignIndividual>
          </owl:hasValue>
        </owl:Restriction>
      </rdfs:subClassOf>
    </owl:Class>
```

where in the “activities” ontology *surfing* is defined to be an individual of the class *SportsActivity*.

```
    (activities)
    <activities:Surfing rdf:ID="surfing"/>
```

A link property can be defined as the inverse of another link property. For example, *link inversion* would allow to define a *WaterSport* activity a *SportsActivity* that is offered at a *BeachDestination*:

```
    (activities)
    <owl:Class rdf:ID="WaterSport">
      <rdfs:subClassOf>
        <owl:Class rdf:about="#Sport"/>
      </rdfs:subClassOf>
      <rdfs:subClassOf>
        <owl:Restriction>
          <owl:onProperty rdf:resource="#isOfferedAt"/>
          <owl:someValuesFrom>
            <ec:ForeignClass rdf:about="&dest;BeachDestination">
              <ec:foreignOntology rdf:resource="&dest;"/>
            </ec:ForeignClass>
          </owl:someValuesFrom>
        </owl:Restriction>
      </rdfs:subclassOf>
    </owl:Class>

    (activities)
    <ec:LinkProperty rdf:ID="isOfferedAt">
      <ec:foreignOntology rdf:resource="&dest;"/>
      <owl:inverseOf rdf:resource="&dest;offersActivity"/>
    </ec:LinkProperty>
```

A link property may be defined as a sub-property of another link property; e.g. we can define *offersSportActivity* as a sub-property of *offersActivity*:

```
    (destinations)
    <ec:LinkProperty rdf:ID="offersSportActivity">
      <ec:foreignOntology rdf:resource="&activities;"/>
      <rdfs:subPropertyOf rdf:about="#offersActivity"/>
    </ec:LinkProperty>
```

Obviously, a link property cannot be declared as a sub-property of an object or datatype property, nor a sub-property of a link relation with a different foreign ontology.

11.2.2 Axioms and Facts in an $\mathcal{E}$-Connected Ontology

An $\mathcal{E}$-Connected ontology is a sequence of axioms and facts : logical sentences that allow to make assertions about the domain. For example, as in OWL, we can use an axiom to assert that *GraduateStudent* is a subclass of *Student* and a fact to state that *john* is *enrolledIn* the *WebTechnologies* course.

In $\mathcal{E}$-Connected ontologies it is also possible to use a fact to instantiate a link property. For example, we can assert that *SaintThomasIsland* is an instance of *CaribbeanDestination* and that it offers the *surfing* activity:

```
(destinations)
<rdf:Description rdf:about="#SaintThomasIsland">
  <rdf:type>
    <owl:Class rdf:about="#CaribbeanDestination"/>
  </rdf:type>
  <offersActivity>
    <ec:ForeignIndividual rdf:about="&activities;#surfing">
      <ec:foreignOntology rdf:resource="&activities;"/>
    </ec:ForeignIndividual>
  </offersActivity>
</rdf:Description>
```

The $\mathcal{E}$-Connections framework imposes some restrictions to axioms and facts. For example, a class cannot be declared in an ontology as a subclass of a class declared in a foreign ontology in the combination. A property (object, datatype or link) cannot be declared as sub-relation of a foreign property; an individual cannot be declared as an instance of a foreign class, and a pair of individuals cannot instantiate a foreign property.

$\mathcal{E}$-Connections also constrain the use of URIs. In OWL-DL, a URI cannot be used, for example, both as a class and a datatype, or as an object property and a datatype property. In an $\mathcal{E}$-Connected ontology, a set of additional restrictions must be imposed, namely a URI cannot be used "locally" in two different component ontologies.

11.2.3 What Is a $\mathcal{E}$-Connection in a Semantic Web Context?

An $\mathcal{E}$-Connection is a set of $\mathcal{E}$-Connected ontologies. However, this definition can be ambiguous, since it may not be apparent at the first sight to *which* set we are referring to.

For example, let us consider again the "tourism" domain. At first sight, one would say that we have a single $\mathcal{E}$-Connection, namely, the one composed by the ontologies: "destinations", "activities", "accommodations", "people". However, this is not strictly correct.

Suppose that the "people" ontology contains an explicit contradiction. Assume we load the "accommodations" ontology in a reasoner and try to check its consistency; what should the reasoner *do*? The "accommodations" ontology contains no link property declarations, i.e., it is an "ordinary" OWL ontology. In this case, an $\mathcal{E}$-Connections aware reasoner should *only* check the consistency of that ontology, and ignore the rest, i.e., it should not report the inconsistency in the "people" ontology.

Given an $\mathcal{E}$-Connected ontology, a reasoner should consider *only* all the $\mathcal{E}$-Connected ontologies in its transitive closure under link references. For example, the "destinations" ontology defines a link property *providesAccommodation* to the "accommodations" ontology and a link *offersActivity* to the "activities" ontology. Since "accommodations" does not contain any link property (it is an ordinary OWL ontology) and "activities" only includes the link property *isOfferedAt* to "destinations" again, the reasoner would consider the following set of $\mathcal{E}$-Connected ontologies:

$$K_{destinations} = \{destinations, accommodations, activities\}$$

We say that $K_{destinations}$ is the $\mathcal{E}$-Connection *induced by* the *destinations* ontology. Looking at the link references between the different ontologies of the example, it is easy to see that:

$$\begin{aligned} K_{accommodations} &= \{accommodations\} \\ K_{activities} &= \{activities, destinations, accommodation\} \\ K_{people} &= \{people, destinations, accommodations, activities\} \end{aligned}$$

An OWL ontology O can be seen as an $\mathcal{E}$-Connected ontology which induces an $\mathcal{E}$-Connection with O as its only component.

11.3 Modeling with $\mathcal{E}$-Connections

$\mathcal{E}$-Connections can be used, as in the example on tourism, for integrating existing ontologies, which describe different application domains, by adding knowledge about how these domains are related to each other.

In this section, we illustrate how to use $\mathcal{E}$-Connections the other way round, namely for decomposing a knowledge base into smaller, connected ontologies. For details about this application, we refer the interested reader to [11, 10].

As an example, let us consider the ontology used in the OWL documentation: the Wine Ontology [28]. This ontology describes different kinds of wines according to various criteria, like the area they are produced in, the kinds of grapes they contain, their flavor and color, etc. For example, a "Cabernet Franc Wine" is defined to be a dry, red wine, with moderate flavor and medium body and which is made with Cabernet Franc grapes[3]:

$$\begin{aligned} \mathit{CabernetFranc} &\equiv \mathit{Wine} \sqcap \exists \mathit{madeFromGrape}.\{\mathit{CabernetFrancGrape}\} \sqcap \\ &\sqcap \leq 1 \mathit{madeFromGrape} \\ \mathit{CabernetFranc} &\sqsubseteq \exists \mathit{hasColor}.\{\mathit{Red}\} \sqcap \exists \mathit{hasFlavor}.\{\mathit{Moderate}\} \\ &\sqcap \exists \mathit{hasBody}.\{\mathit{Medium}\} \end{aligned}$$

A Bordeaux is defined to be a wine produced in the Bordeaux area[4]:

$$\mathit{Bordeaux} \equiv \mathit{Wine} \sqcap \exists \mathit{locatedIn}.\{\mathit{BordeauxRegion}\}$$

Note that the Wine Ontology does not contain information about wines only, but also information about regions, wineries, colors, grapes, and so on. This illustrates a

[3] In this section, instead of using the RDF/XML notation as in Section 2 we will use for clarity and brevity standard DL notation. The equivalence between this notation and abstract syntax is summarized in Tables 2 and 3.

[4] In both examples the braces represent nominals.

general feature of OWL ontologies: although they usually refer to a core application domain, they also contain "side" information about other secondary domains.

This modeling paradigm is not only characteristic of small and medium sized ontologies, but also occurs in large, high-quality knowledge bases, written by groups of experts. A prominent example is the NCI (National Cancer Institute) ontology [12], a huge ontology describing the cancer domain. The NCI ontology is mainly focused on genes, but it also contains some information about many other domains, like professional organizations, funding, research programs, etc.

In all these cases, it is more natural to represent each application domain in a different $\mathcal{E}$-Connected ontology, where link properties are used in the component KBs whenever information from a different ontology in the combination is required. The component ontologies in an $\mathcal{E}$-Connection are mostly self-contained in the sense that they only contain information about a single "topic", and are loosely coupled, since the coupling between $\mathcal{E}$-Connected ontologies can only be achieved through restrictions on link properties.

In the case of the Wine Ontology, the combined KB is composed of six ontologies, dealing with grapes, wine descriptors (color, flavor,...), regions, wineries, years, and wines respectively[5].

When transforming a DL knowledge base into an $\mathcal{E}$-Connected KB, many object properties in the original ontology become link properties in the $\mathcal{E}$-Connected KB. For example, in the definition of Cabernet Franc wines, the object property $madeFromGrape$ becomes a link property from the wine ontology to the grapes ontology, while $hasColor$, $hasFlavor$ and $hasBody$ connect the wine ontology and the wine descriptors ontology. Obviously, this effect depends on the number of $\mathcal{E}$-Connected ontologies.

The use of $\mathcal{E}$-Connections for both integration and decomposition of OWL knowledge bases suggests a new modeling methodology, applicable to knowledge engineering with Description Logics in general, and to the Semantic Web in particular. The core idea is to keep ontologies small and disjoint and to use these ontologies as reusable units that can be combined in various ways using $\mathcal{E}$-Connections depending on the modeler's needs.

It is worth emphasizing here that $\mathcal{E}$-Connections are *not* a suitable technique for combining ontologies dealing with highly overlapping domains, which prevents its use in some important Knowledge Engineering applications, such as ontology refinement. For example, if we developed a new ontology on grapes with richer descriptions about a certain kind of grape, we would not be able to connect it to the old one using our technique. However, $\mathcal{E}$-Connections were not designed for such a purpose.

11.4 Tool Support

In this section, we discuss the key issues to be addressed for providing tool and application support for $\mathcal{E}$-Connections, and we describe our implementation of an

[5] In this section, we are using a version of the Wine Ontology that does not import the Food Ontology.

$\mathcal{E}$-Connection aware infrastructure that extends the OWL-API [6] and is integrated to the ontology editor Swoop [23].[6]

11.4.1 Requirements

Basic Functionality

A basic implementation for $\mathcal{E}$-Connections must perform similar tasks as an OWL implementation itself:

- Serializing: Producing the extended OWL concrete syntax, introduced in Section 2, from some internal representation.
- Modeling: Providing a suitable internal representation (model) for $\mathcal{E}$-Connected ontologies.
- Parsing: Building an internal representation from a document in the concrete syntax.

This basic functionality must be provided by Semantic Web high-level programming interfaces, such as the OWL-API.

Ontology browsers and editors must provide additional rendering and editing functionality. Perhaps, the most important requirement for an $\mathcal{E}$-Connections aware ontology editor is the ability to deal effectively with multiple ontologies, which implies, for example, the ability to load, save and manipulate several ontologies simultaneously, as well as the ability to easily navigate from one to another.

Ontology editors must also be able to provide browsing and editing capabilities for the extended RDF/XML and abstract syntaxes[7] and support for the simultaneous use of imports and $\mathcal{E}$-Connections.

Combining Imports and $\mathcal{E}$-Connections

The combined use of $\mathcal{E}$-Connections and *owl:imports* raises a number of difficulties. For example, suppose a simple case in which we have two connected ontologies: the ontology O_A about "people" and the ontology O_B about "animals". Suppose that in O_A there are link properties connecting O_A to O_B and vice-versa. Assume that the modeler decides at some point that persons should be described as a subclass of animals and that the application which is using the connected ontologies relies on such a decision. Consequently, the modeler makes O_A import O_B. The following issue immediately arises: what should a tool automatically *do* in such a case?

In principle, an editor should automatically modify the ontology O_A, and leave the ontology O_B unaltered in the workspace, in order to comply with the asymmetric nature of the *owl:imports* construct. Then, the tool should transform, in O_A, all the link properties from O_A to O_B and from O_B to O_A [8] into ordinary object properties.

[6] Swoop is available for download at `http://www.mindswap.org/2004/SWOOP`

[7] The extended abstract syntax for $\mathcal{E}$-Connections will be presented in next section.

[8] Note that those are imported.

In other words, O_A is transformed into a "plain" OWL ontology, which treats the domains of people and animals as a single one.

However, those modifications leave the ontology O_B, which is still $\mathcal{E}$-Connected to the (modified) O_A ontology, in a pretty much non-sensical state, since O_B would be $\mathcal{E}$-Connected to the "union" of O_A and itself. What should happen next? Clearly, the $\mathcal{E}$-Connection has been "broken" by the *owl:imports* statement, since, as we will discuss later, there is a violation in the restrictions on the usage of URIs within an $\mathcal{E}$-Connection.

In such a situation, "merging" O_A and O_B turns out to be the most plausible solution. A *merge* would enforce that all the link properties pointing to O_B will now point to O_A (which is importing O_B) and all the link properties *from* O_B will become object properties. These operations will "disconnect" O_B from the combination.

11.4.2 Implementation

Extending the OWL-API

We have extended Manchester's OWL-API in order to provide a high-level programmatic interface for parsing, serializing and manipulating $\mathcal{E}$-Connected ontologies.

The OWL-API RDF parser and internal model have been extended to deal with the new constructs. Link properties in an $\mathcal{E}$-Connected ontology are internally represented as object properties that additionally specify the URI of its target ontology. On the other hand, $\mathcal{E}$-Connected ontologies extend OWL ontologies with functionality for retrieving link properties and foreign entities (classes, properties and individuals). In order to manipulate the structures in the internal representation, we have provided functionality for adding/removing a foreign entity to/from an $\mathcal{E}$-Connected ontology, and to set the target ontology of a link property. The functionality for adding/removing link properties is provided by reusing the corresponding functionality for object properties.

Extending Swoop

Swoop [23] is a Web Ontology browser and editor.

Swoop assumes the use of multiple ontologies and supports this use in a number of ways. Being a multiple ontology engineering environment, Swoop is an ideal testbed for experimenting with $\mathcal{E}$-Connections in the Semantic Web context.

Swoop uses the new functionality added to the OWL-API in order to provide basic rendering and editing support for $\mathcal{E}$-Connections. As in the case of the OWL-API, we have tried to keep the main UI and design decisions in Swoop and to incorporate the required functionality with a minimal extension of the code.

The extension of the OWL-API automatically provides the functionality for loading and saving $\mathcal{E}$-Connected ontologies in the extended RDF/XML syntax presented in Section 2. When loading an $\mathcal{E}$-Connected ontology, link properties and foreign entities are distinguished in the UI from the rest of OWL entities.

All these new elements are hyperlinked; thus, for example, if the user clicks on the URI of a foreign ontology, the ontology will be loaded in the workspace. The idea is to provide easy and intuitive navigation through the ontologies in an $\mathcal{E}$-Connection.

Swoop provides basic tool support for creating and editing $\mathcal{E}$-Connected ontologies. Link properties can be added to the ontology and it is possible to define restrictions on them. Incorrect editing is prevented when possible: link properties cannot be made transitive or symmetric and restrictions on link properties can only be applied to classes/individuals in its *target* ontology.

Finally, Swoop provides a graph layout for visualizing the connections between components in an $\mathcal{E}$-Connection. Given a selected ontology in the workspace, Swoop provides means for displaying its induced $\mathcal{E}$-Connection.

11.5 An Extension of OWL-DL

11.5.1 Abstract Syntax

The syntax presented in this section is an extension of the normative OWL abstract syntax, as described in the OWL abstract syntax and semantics recommendation [26], and we use the same Extended BNF syntax as in the normative document.

An $\mathcal{E}$-Connected ontology K contains a sequence of annotations, axioms and facts. Annotations, as in OWL, can be used to record authorship and other information associated with the ontology, including imports.

E-ConnectedOntology :: =
    'E-ConnectedOntology(' [**ontologyID**] { **directive** } ')'
directive :: = 'Annotation(' **ontologyPropertyID ontologyID** ')'
    | 'Annotation(' **annotationPropertyID URIreference** ')'
    | 'Annotation(' **annotationPropertyID dataLiteral** ')'
    | 'Annotation(' **annotationPropertyID individual** ')'
    | **axiom**
    | **fact**

$\mathcal{E}$-Connected ontologies are referenced using a URI. $\mathcal{E}$-Connected ontologies contain information about the same kind of entities as OWL ontologies (classes, object properties, etc.), but they also contain information about link properties. Link properties are also denoted by URI references.

linkID :: = URIreference

In order to ensure the separation of vocabularies, a URI-reference cannot be both a linkID, an individual-valued property ID, a datatype property ID, an annotation property ID or an ontology property ID in an $\mathcal{E}$-Connected ontology. Intuitively, while individual-valued properties relate individuals to other individuals in the same ontology, link properties relate individuals corresponding to different interpretation domains. Thus, link properties act as a bridge between different ontologies, which remain separate, and keep their own identity.

An OWL ontology in the abstract syntax contains a sequence of axioms and facts. In order to provide support for $\mathcal{E}$-Connections on the Semantic Web, we propose a syntactic and semantic extension of OWL-DL with new kinds of axioms and facts. Every OWL ontology can be seen as an $\mathcal{E}$-Connected ontology in which no link properties have been declared.

Axioms

In a $\mathcal{E}$-Connected ontology, link properties can be defined using the following axiom:

axiom ::= ‘Link(’ **linkID**[‘Deprecated’] { **annotation** }
    ‘foreignOntology(’ **OntologyID**’)’
    { ‘super(’ linkID ‘)’ }
    { ‘domain(’ **description** ‘)’ }
    { ‘range(’ **foreignDescription** ‘)’ }
    [‘inverseOf(’ **linkID** ‘)’]
    [‘Functional’ | ‘InverseFunctional’]

Link properties used in an abstract syntax ontology *must* be declared, and hence need an axiom. A link property cannot be declared twice as referring to different ontologies.

Link properties can be made functional or inverse functional and can be given global domains and ranges. As opposed to object properties, link properties cannot be made transitive or symmetric.

Link properties can be equivalent to or sub-properties of others. Of course, in order for these axioms to model useful information, the link properties that are related through equivalence or subsumption should refer to the same foreign ontology.

axiom ::= ‘EquivalentProperties(’ **linkID linkID** { **linkID** } ‘)’
    | ‘SubPropertyOf(’ **linkID linkID** ‘)’

In $\mathcal{E}$-Connections the coupling between the ontologies is achieved through restrictions on link properties. As in OWL, universal (*allValuesFrom*), existential (*someValuesFrom*) and value (*hasValue*) restrictions can be defined.

restriction::= ‘Restriction(’**linkID**
    linkRestrictionComponent { **linkRestrictionComponent** } ’)’
linkRestrictionComponent::= ‘allValuesFrom(’ **foreignDescription** ‘)’
    | ‘someValuesFrom(’ **foreignDescription** ‘)’
    | ‘value(ForeignIndividual(’ **individualID** ‘))’
    | **cardinality**

Range axioms and restrictions on link properties are referred to foreign class descriptions. Foreign classes are classes that, though used in a certain $\mathcal{E}$-Connected ontology, correspond to a different ontology in the combined knowledge base. If a foreign description is used in a range axiom or in a restriction corresponding to a link property, it will always be interpreted in the domain of the target ontology of the link property.

foreignDescription ::= ‘ForeignClass(’ **description** ‘)’

Facts

Our proposal extends the OWL-DL facts by adding the following production rule to the normative OWL abstract syntax:

value ::= ‘value(’ ForeignIndividual(**linkID** individualID ’)’
These facts allow to instantiate link properties with named individuals.

11.5.2 Direct Model-Theoretic Semantics

This section provides a model-theoretic semantics to $\mathcal{E}$-Connected ontologies written in the abstract syntax.

As in OWL, the definition of the semantics starts with the notion of a *combined vocabulary.*

Definition 1. *A* ***combined OWL vocabulary*** *V consists of a set V_L of literals and the following sets of URI references: For $i = 1, ..., n$, V_{C_i} are sets of class names, each of which contains owl:Thing and owl:Nothing; V_{I_i} are sets of individual names; V_{DP_i}, V_{IP_i} and V_{AP_i} are sets of datatype, object and annotation property names respectively, where each V_{AP_i} contains owl:versionInfo, rdfs:label, rdfs:comment, isDefinedBy, seeAlso; V_D is the set of datatype names, which also contains the URI references for the built-in OWL datatypes and rdfs:Literal; V_O the set of ontology names and V_{OP} the set of URI references for the built-in ontology properties; finally, for $i, j = 1, ..., n$ with $i \neq j$, $\mathcal{E}_{ij}$ are sets of URI references denoting link properties.*

In any vocabulary, the $(V_{C_i} - \{owl : Thing, owl : Nothing\})$ are pair-wise disjoint, and disjoint with V_D. Also, for each $i, j = 1, ..., n$ with $i \neq j$, the V_{DP_i},V_{IP_i},$(V_{AP_i} - \{owl : versionInfo, rdfs : label, rdfs : comment\})$,$V_{OP}$, $\mathcal{E}_{ij}$ are all pair-wise disjoint. ◇

Given an $\mathcal{E}$-Connected ontology, the vocabulary *must* include all the URI references and literals utilized in each of the ontologies, as well as those used in ontologies that are imported or referenced through a link property by any of the component ontologies, but can contain other URI references and literals as well.

As in OWL, a datatype d is characterized by a lexical space, $L(d)$, which is a set of Unicode strings; a value space, $V(d)$; and a total mapping $L_2V(d)$ from the lexical space to the value space.

Definition 2. *A* ***datatype map*** *D is a partial mapping from URI references to datatypes that maps xsd:string and xsd:integer to the appropriate XML Schema datatypes.* ◇

The original $\mathcal{E}$-Connections proposal did not explicitly mention datatypes; however, these trivially fit within the framework and we will explicitly include them here, since they are an important component of OWL.

The model-theoretic semantics is provided by the notion of a *combined interpretation*.

Definition 3. *Let D be a datatype map. A* ***combined OWL interpretation*** *with respect to D with combined vocabulary V is a tuple of the form:*

$$I = (R, (R_i)_{1\le i\le n}, (EC_i)_{1\le i\le n}, (ER_i)_{1\le i\le n}, L, (S_i)_{1\le i\le n}, ED, LV, N),$$

where (with $\mathcal{P}$ being the powerset operator):

- *R the set of resources of the interpretation is a non-empty set*
- *$LV \subset R$, the set of literal values of I, contains the set of Unicode strings, the set of pairs of Unicode strings and language tags, and the value spaces for each datatype in D*
- *$R_i = O_i \cup LV$, where O_i is non-empty, disjoint from LV and disjoint from $O_j, \forall j = 1, ..., n; i \neq j$*
- $EC_i : V_{C_i} \to \mathcal{P}(O_i)$
- $ED : V_D \to \mathcal{P}(LV)$
- $ER_i : V_{IP_i} \to \mathcal{P}(O_i \times O_i)$
- $ER_i : V_{DP_i} \to \mathcal{P}(O_i \times LV)$
- $ER_i : \mathcal{E}_{ij} \to \mathcal{P}(O_i \times O_j)$
- *$L : TL \to LV$, where TL is the set of typed literals in V_L*
- $ER_i : V_{AP_i} \to \mathcal{P}(R_i \times R_i)$
- $ER_i : V_{OP} \to \mathcal{P}(R_i \times R_i)$
- $S_i : V_{I_i} \to O_i$
- *S_i : $V_{I_i} \cup V_{C_i} \cup V_D \cup V_{DP_i} \cup V_{IP_i} \cup V_{AP_i} \cup \mathcal{E}_{ij} \cup \{$ owl:Ontology$\} \to N$, where $N \subset R$[9] is disjoint with LV and with each of the $O_i, \forall i = 1, ..., n$*
- $EC_i(owl : Thing) = O_i \subseteq R$
- $EC_i(owl : Nothing) = \emptyset$
- $EC_i(rdfs : Literal) = LV$
- *If $D(d') = d$ then $ED(d') = V(d)$*
- *If $D(d') = d$, then $L(`v'^{\wedge\wedge}d') = L_2V(d)(v)$*
- *If $D(d') = d$ and $v \notin L(d)$, then $L(`v'^{\wedge\wedge}d') \in R - LV$* $\diamond$

In a combined OWL interpretation, the functions $(EC_i)_{1\le i\le n}$ provide logical meaning for URI references used as classes in the $\mathcal{E}$-Connected ontology K_i. The functions $(ER_i)_{1\le i\le n}$ assign logical meaning to the URIs in the ontologies $(K_i)_{1\le i\le n}$ that are used as OWL properties or links[10]. In a combined interpretation the abstract interpretation domain is partitioned into n disjoint parts, each of which corresponding to a different component ontology. Link properties are interpreted through $(ER_i)_{1\le i\le n}$ as pairs of elements corresponding to different parts of the abstract logical domain.

[9] Vocabulary Names.

[10] As in OWL, the property *rdf:type* is added to the annotation properties in order to provide meaning for deprecation.

Table 11.1. Extension of EC_i

Abstract Syntax	Interpretation (value of EC_i)
restriction(l allValuesFrom(ForeignClass(c)))	$\{x \in O_i \mid (x, y) \in ER_i(l) \Rightarrow y \in EC_j(c)\}$
restriction(l someValuesFrom(ForeignClass(c)))	$\{x \in O_i \mid (x, y) \in ER_i(l) \wedge y \in EC_j(c)\}$
restriction(l value(ForeignIndividual(id)))	$\{x \in O_i \mid (x, S_j(id)) \in ER_i(l)\}$
restriction(l minCardinality(n))	$\{x \in O_i \mid \#(\{y \in O_j : (x, y) \in ER_i(l)\}) \geq n\}$
restriction(l maxCardinality(n))	$\{x \in O_i \mid \#(\{y \in O_j : (x, y) \in ER_i(l)\}) \leq n\}$
restriction(l Cardinality(n))	$\{x \in O_i \mid \#(\{y \in O_j : (x, y) \in ER_i(l)\}) = n\}$

Table 11.2. Interpretation of Axioms and Facts

Directive	Conditions on Interpretations
Link(l foreignOntology(Ont_j) annotation(p_1 o_1) ... ... annotation(p_k o_k) super(s_1) ... super(s_n)) domain(d_1)... domain(d_n) range(r_1) ... range(r_n) [inverseOf(m)] [Functional] [InverseFunctional])	 $ER_i(l) \subseteq O_i \times O_j$ $S_i(l) \in EC_i(annotation(p_1\ o_1))...$ $...S_i(l) \in EC_i(annotation(p_k\ o_k))$ $ER_i(l) \subseteq ER_i(s_1) \cap ... \cap ER_i(s_n)$ $ER_i(l) \subseteq EC_i(d_1) \times O_j \cap ... \cap EC_i(d_n) \times O_j$ $ER_i(l) \subseteq O_i \times EC_j(r_1) \cap ... \cap O_i \times EC_j(r_n)$ $ER_i(l) = (ER_j(m))^-$ $\forall x \in O_i\ \forall y, z \in O_j, (ER_i(l))(x, y)$ $\wedge (ER_i(l))(x, z) \rightarrow y = z$ $\forall y, z \in O_i, \forall x \in O_j, (ER_i(l))(y, x)$ $\wedge (ER_i(l))(z, x) \rightarrow y = z$
EquivalentProperties($l_1 ... l_n$)	$ER_i(l_j) = ER_i(l_k) \forall 1 \leq j \leq k \leq n$
SubpropertyOf(l_1, l_2)	$ER_i(l_1) \subseteq ER_i(l_2)$

The function L provides meaning for typed literals and the function ED to the datatypes used in the $\mathcal{E}$-Connection. Note that, as opposed to the abstract interpretation domain, the domain in which datatypes and literals are interpreted is not partitioned.

The functions $(S_i)_{1 \leq i \leq n}$ provide meaning to OWL individuals. Analogously to OWL, these functions are extended to plain literals in V_L by mapping them onto themselves. Note that, if the same literal "l" is used in different component ontologies, say i, j, the functions S_i, S_j will map it to the same value in LV, i.e. $S_i(l) = S_j(l) = l \in LV$.

The functions EC_i are extended to class descriptions, individuals and annotations an in the OWL specification [26]; the additional constructs are interpreted according to Table 11.1, where l is a link property declared in K_i with foreign ontology K_j.

A combined OWL interpretation I satisfies axioms and facts as given in the OWL specifications [26] and in Table 11.2.

We now define precisely the notion of the $\mathcal{E}$-Connection induced by an ontology, introduced informally in Section 2.3.

Definition 4. *Let K_1 be an $\mathcal{E}$-Connected ontology. The $\mathcal{E}$-Connection* ***induced by*** *of K_1 is defined to be the following set of $\mathcal{E}$-Connected ontologies:*

$$K = (K_1, ..., K_n) = clos(K_1)$$

where the set $clos(K_1)$ is inductively defined as follows:

- *$K_1 \in clos(K_1)$.*
- *If K' belongs to $clos(K_1)$ and there is a link property with source K' and target K'', then $K'' \in clos(K_1)$.* ◇

Note that an $\mathcal{E}$-Connected ontology K_1 is an "ordinary" OWL ontology, as specified in the normative documents if and only if the $\mathcal{E}$-Connection induced by K_1 is precisely itself, i.e. $K = (K_1)$.

Since a URI can be used in an $\mathcal{E}$-Connected ontology either as *local* or *foreign*, we need to provide a formal distinction between both cases. For such a purpose, we introduce the notion of a URI to belong to an $\mathcal{E}$-Connected ontology as follows:

Definition 5. *Let $K = (K_1, ..., K_n)$ be the $\mathcal{E}$-Connection induced by K_1. We say that a URI reference u* ***belongs to*** *the $\mathcal{E}$-Connected ontology K_i if either of the following conditions holds:*

- *It is used in K_i or in an ontology imported by K_i, but not in the context of a restriction on a link property.*
- *It is used in $K_j, j \neq i$ in the context of a restriction on a link property with foreign ontology K_i.* ◇

For example, assume that the URI *foo:Publication* is used in an $\mathcal{E}$-Connected ontology O_1 in the context of the following restriction: restriction(l ForeignClass (foo:Publication)), where l is defined to be a link property with foreign ontology O_2; then, the URI would *belong to* the ontology O_2 and would *necessarily* need to be declared as a class in O_2.

The semantics of an $\mathcal{E}$-Connection is given as follows:

Definition 6. *Let D be a datatype map, K_1 be an $\mathcal{E}$-Connected ontology, and $K = (K_1, ..., K_n)$ the $\mathcal{E}$-Connection induced by K_1 . A combined OWL interpretation I with respect to D with vocabulary V,* ***satisfies*** *K_1 (denoted by $I \models K_1$) iff:*

1. *Each URI reference* ***belonging to*** *K_i, used as a classID (datatypeID, individualID, data-valued property ID, annotation property ID, annotation ID, ontology ID) belongs to a* single *component ontology K_i and is contained in $V_{C_i}(V_D, V_{I_i}, V_{DP_i}, V_{IP_i}, V_{AP_i}, V_O$, respectively).*
2. *Each literal in K_i belongs to V_L.*

3. *Each URI reference in K_i used as a linkID, with foreign ontology K_j, is in $\mathcal{E}_{ij}$.*
4. *I satisfies each directive in K_i, except for ontology annotations.*
5. *For each $i = 1, ..., n$ there is some $o_i \in N \subset R$ with $(o_i, S_i(owl : Ontology)) \in ER_i(rdf : type)$ such that for each ontology annotation $Annotation(p\ v)$, $(o_i, S_i(v)) \in ER_i(p)$ and if the component ontology K_i has name n_i, then $S_i(n_i) = o_i$.*
6. *I satisfies each $\mathcal{E}$-Connected ontology in K.*
7. *I satisfies each ontology mentioned in an owl:imports annotation directive of any K_i.* $\diamond$

At this point, it is worth discussing the semantics we have provided to URIs within an $\mathcal{E}$-Connection.

The meaning of names is a contentious issue in the Semantic Web. Numerous proposals have been given for how to provide meaning for names in the Semantic Web, ranging from a strict localized model-theoretic semantics to proposals for a unified single meaning. Certainly, the latter was the original goal of URIs, as "global" identifiers. However, currently, the meaning of a name in RDF and hence in OWL is relative to a particular RDF graph [7]. In other words, the meaning of the same URI in *other* documents is not considered *at all.* The only way that the OWL standards provide in order to take into account the meaning of a URI in a different ontology is to *import* that ontology. When an ontology imports another one, identical URIs are *merged*; otherwise, the meaning of a URI is entirely "local" to an RDF/OWL document.

In the framework of $\mathcal{E}$-Connections, we provide a *stronger* meaning to URIs. Roughly, we prevent the use of the same global name (URI reference) to denote *different* things within an $\mathcal{E}$-Connection.

Recall also that, although each ontology within an $\mathcal{E}$-Connection is interpreted in a different domain, datatypes in every ontology are interpreted in the *same* domain. As opposed to the case of object domains, there is no reason to partition the datatype domain. Note that the original $\mathcal{E}$-Connections framework as presented in [25] does not consider datatype theories, nor we considered datatypes in the reasoning algorithms presented in [9]. However, our approach in this paper concerning datatypes does not affect the results in [25] and allows for a straightforward extension of the algorithms in [9].

The main reasoning services are, as in OWL, consistency and entailment.

Definition 7. *An $\mathcal{E}$-Connected ontology K_1 is* ***consistent*** *with respect to a datatype map D (denoted by $I \models_D K$ iff there is some combined interpretation I with respect to D such that I satisfies K_1.* $\diamond$

Definition 8. *An $\mathcal{E}$-Connected ontology K_1* ***entails*** *an $\mathcal{E}$-Connected ontology K_2 with respect to a datatype map D, denoted by $K_1 \models_D K_2$ iff every combined interpretation that satisfies K_1 with respect to D also satisfies K_2 with respect to D.* $\diamond$

The following results are a straightforward consequence of the definitions of entailment and satisfaction under a combined interpretation:

Consequence 1. *Let K_1 be an $\mathcal{E}$-Connected ontology and $K = (K_1, ..., K_n)$ be the $\mathcal{E}$-Connection induced by K_1. Then $K_1 \models_D K_j$, $\forall j = 1, ..., n$.*

Consequence 2. *The $\mathcal{E}$-Connection $K = (K_1, ..., K_n)$ induced by K_1 entails the $\mathcal{E}$-Connection $O = (O_1, ..., O_m)$ induced by O_1 with respect to a datatype map D ($K \models_D O$) iff $K_1 \models_D O_1$.*

These results show that a $\mathcal{E}$-Connection is identified in a Semantic Web context by its "generating" $\mathcal{E}$-Connected ontology.

11.6 Reasoning

11.6.1 Reasoning in OWL

OWL-DL and OWL-Lite can be seen as expressive Description Logics, with an ontology being equivalent to a Description Logics knowledge base. Among the myriad of very expressive DLs, the $\mathcal{SH}$ family of logics plays a prominent role [19]. All modern, highly optimized DL reasoners , such as FaCT [15], RACER [31] and Pellet[27] have been designed for these logics.

In order to obtain a suitable balance between computational properties and expressivity, the design of the DL-based species of OWL has been grounded on the

Table 11.3. The $\mathcal{SH}$ family of Description Logics

Construct Name	DL Syntax	OWL Syntax	Logic
Atomic Concept	A	A(URI reference)	$\mathcal{S}$
Universal Concept	$\top$	owl:Thing	
Atomic Role	R	R (URI reference)	
Conjunction	$C \sqcap D$	intersectionOf(C,D)	
Disjunction	$C \sqcup D$	unionOf(C,D)	
Negation	$\neg C$	ComplementOf(C)	
Existential Restriction	$\exists R.C$	restriction(R someValuesFrom(C))	
Value Restriction	$\forall R.C$	restriction(R allValuesFrom(C))	
Transitive Role	$Trans(R)$	ObjectProperty(R [Transitive])	
Role Hierarchy	$R \sqsubseteq S$	subPropertyOf(R,S)	$\mathcal{H}$
Inverse Role	$S = R^-$	ObjectProperty(S [inverseOf(R)])	$\mathcal{I}$
Nominals	$\{o_1, ..., o_n\}$	OneOf($o_1, ..., o_n$)	O
Functional Role	$Funct(R)$	ObjectProperty(R [Functional])	$\mathcal{F}$
Functional	$\geq 2R$	restriction(R minCardinality(1))	
Number Restrictions	$\leq 1R$	restriction(R maxCardinality(1))	
Unqualified	$\geq nR$	restriction(R minCardinality(n))	$\mathcal{N}$
Number Restrictions	$\leq nR$	restriction(R maxCardinality(n))	

$\mathcal{SH}$ family of logics. OWL-Lite corresponds to $\mathcal{SHIF}(\mathcal{D})$, while OWL-DL corresponds to $\mathcal{SHOIN}(\mathcal{D})$.

The first algorithm for the logic $\mathcal{SH}$ was presented in [14]. The extension for $\mathcal{SHIF}$ was presented in [21], and qualified number restrictions (an extension of the number restrictions used in OWL) were introduced for the logic $\mathcal{SHIQ}$ in [19]. Nominals and datatypes were presented for the logic $\mathcal{SHOQ}(\mathcal{D})$ in [17]. In [13] it was proved using the tableau systems formalism that satisfiability in $\mathcal{SHIO}$ is can be decided with a tableau algorithm. Finally, the design of a reasoning procedure for $\mathcal{SHOIQ}$ has been achieved recently [22].

11.6.2 A Family of $\mathcal{E}$-Connection Laguages

There are two ways to obtain new $\mathcal{E}$-Connection languages. The first possibility is to vary the set of component logics; the second would be to change the logic of the link properties, i.e., to vary the set of operators that can be applied on link properties. Different choices in the component logics and in the logic of the link properties will yield, in general, different combination languages, with different expressivity and computational properties.

A family of $\mathcal{E}$-Connection languages can be specified by fixing the component logics and varying the logic of the link properties. In this paper, we have tacitly specified, in the abstract syntax and semantics section the $\mathcal{E}$-Connection language that allows the use all the expressivity of $\mathcal{SHOIN}(\mathcal{D})$, and hence of OWL-DL, in the component ontologies, and the use of inverse link properties, link hierarchies and *someValuesFrom*, *allValuesFrom*, *hasValue* and cardinality restrictions on link properties.

It is not hard to see that the first requirement for reasoning on an $\mathcal{E}$-Connection language is the ability to reason independently on each of its component logics. Our aim is to show that it is possible to design and implement practical tableau-based algorithms for $\mathcal{E}$-Connections on top of existing DL reasoners. Therefore, in this paper, we will focus on the family of combination languages involving OWL-DL—that is, $\mathcal{SHOIN}(\mathcal{D})$—ontologies.

Once the component logics have been identified, it remains to investigate the expressivity that should be allowed on link properties. Due to the presence of inverses in this language, cardinality restrictions on link properties allow to transfer nominals in its full generality from one component to another [25], which would invalidate the separation of domains and hence "break" the $\mathcal{E}$-Connection.[11] Note that the problem we just described is related to the ability to transfer individuals between components, which violates the principles of $\mathcal{E}$-Conenctions; it is not related to decidability in the case the $\mathcal{E}$-connection only combines $\mathcal{SHOIN}$ is always decidable.

However, it is worth emphasizing here that, in many special cases, we can still process combined OWL-DL ontologies, even if number restrictions and inverses on link properties, as well as nominals, are used in the combination. Indeed such an $\mathcal{E}$-Connected KB $K = (K_1, ..., K_n)$ can be processed as long as, for every possible pair

[11] Obviously, if none of the component logics contained nominals, it would be safe to use the full expressivity on the link properties.

of component ontologies K_i, K_j for $i, j \in \{1, ..., n\}, i \neq j$, the following conditions *do not hold simultaneously*:

1. Some K_i contains nominals.
2. An inverse of a link property from K_i to K_j or vice-versa is used.
3. Number restrictions on links from K_i to K_j or vice-versa are used.

Of course, in order to be able to handle arbitrary combined ontologies, we need to further restrict the expressivity allowed on link properties. In the sections that follow, we will discuss how to handle algorithmically the $\mathcal{E}$-Connections verifying the conditions above.

11.6.3 Tableau Algorithms for $\mathcal{E}$-Connections of Description Logics

In this section, we provide a general intuition on how the reasoning algorithms we have developed for $\mathcal{E}$-Connections work. We will not include here a detailed presentation and refer the interested reader to [9] for a detailed discussion. We assume that the reader is familiar with tableaux algorithms for DLs.

Modern DL reasoners, like FaCT, RACER and Pellet implement the tableaux method [4] for solving the main reasoning tasks in expressive Description Logics.

Tableau algorithms are focused on satisfiability; other reasoning problems, like subsumption, or entailment (the main inference problem in OWL) can be solved by first reducing them to satisfiability [3].

In order to check satisfiability of a given concept C_0 w.r.t. a knowledge base Σ, a tableau algorithm tries to construct a common model of C_0 and Σ. If it succeeds, then the algorithm determines that C_0 is satisfiable w.r.t. Σ, and unsatisfiable otherwise.

The main elements that specify a tableau algorithm are [5]:

- An underlying data structure, called the completion graph.
- A set of expansion rules.
- A blocking condition, for ensuring termination.
- A set of clash-triggers, to detect logical contradictions (clashes).

Completion graphs are finite, labeled, directed graphs, which roughly correspond to abstractions of possible models for C and Σ.

Each node and edge in a completion graph $\mathcal{G}$ is labeled with a set of concepts and a set of roles respectively. To decide the satisfiability of C w.r.t. Σ, the algorithm generates an initial graph $\mathcal{G}$, constructed from C and Σ and repeatedly applies the expansion rules until a clash (i.e. a contradiction) is detected in the label of a node, or until a clash-free graph is found to which no more rules are applicable. The application of a rule may add new concepts to the label of a node, trigger the generation of a new node or cause two different nodes to merge.

Tableau algorithms for expressive DLs are *non-deterministic* in the sense that there might exist completion rules that yield more than one possible outcome. A tableau algorithm will return “satisfiable” iff there exists at least one way to apply the non-deterministic rules such that a clash-free graph is obtained, to which no rules are applicable.

Termination is guaranteed through *blocking*: halting the expansion process when a "cycle" is detected [4]. When the algorithm detects that a path in the graph will be expanded forever without encountering a contradiction, then the application of the generating rules is blocked, so that no new nodes will be added to that path. There are different kinds of blocking conditions, depending on the presence of inverses and number restrictions in the logic.

The basic strategy to extend a DL tableau algorithm with $\mathcal{E}$-Connections support is based on "coloring" the completion graph [12]. Nodes of different "colors", or sorts, correspond to different domains (ontologies).

The presence of $\mathit{someValuesFrom}$ and $\mathit{minCardinality}$ restrictions on a node label trigger the generation of a successor node of a different "color". The presence of different kinds of nodes in a the graph has several consequences in the way the tableau algorithm works:

- A node may only be blocked by an ancestor node with the same color, and the blocking condition applied to those nodes depends on the logic of the corresponding component ontologies. This implies, for example, that if a certain component ontology does not contain inverse object properties, we can apply subset blocking to its corresponding nodes in the tableau expansion even if *other* component ontologies do contain inverses.
- If a node x is a successor of a node y with a different color, then a special blocking condition is applied to x. Such a blocking condition *only* depends on the logic of the link properties, and not on the logic of the component ontology to which x corresponds. Intuitively, if inverses on link properties are present, then equality blocking is applied; otherwise, subset blocking suffices.
- In Description Logics, inclusions between complex classes C, D are usually transformed into a single class involving a disjunction between D and the negation of C. Such a concept is then added to the label of all the nodes in the graph during the execution of the algorithm. With $\mathcal{E}$-Connections, each of those classes is added only to the labels of nodes corresponding to that component ontology.
- The presence of *maxCardinality restrictions* and nominals (has-Value, oneOf) in node labels may cause the merge of two nodes in the graph. Obviously, only nodes of the same color can ever be merged.

When implementing tableau algorithms for $\mathcal{E}$-Connections as an extension of an OWL reasoner, all these issues have to be thoroughly considered.

11.6.4 Implementation in an OWL Reasoner

We have implemented the tableau algorithms for the $\mathcal{E}$-Connection languages that combine OWL-DL ontologies by either disallowing inverses or number restrictions on the link properties in the OWL reasoner Pellet.

Pellet is a sound and complete tableau reasoner for the description logic $\mathcal{SHOIN}(\mathcal{D})$. Pellet implements the usual suite of optimizations, including lazy

[12] Please, note that our problem has no relation with the graph coloring problem. We just use the term "color" metaphorically to distinguish between different kinds of nodes.

unfolding, absorption, dependency directed backjumping, and semantic branching. It provides support for handling conjunctive ABox queries and incorporates datatype reasoning for the built-in primitive XML Schema datatypes. Pellet is implemented in pure Java and available as open source software.

Pellet has been extended to process $\mathcal{E}$-Connected ontologies, which are implemented as a collection of TBoxes and RBoxes, indexed by the ontology they correspond to. During parsing, each class, object property, datatype property and link property is added to its corresponding component, and after parsing each component of the KB is pre-processed separately.

When performing a satisfiability test, the nodes in the tableau expansion are also labeled with the ontology they refer to. Links are implemented as object properties of a special kind, since they indicate the name of the foreign ontology they point to. When the generating link rules are applied, the ontology label of the new nodes is set to the foreign ontology of the link property involved in the rule application. The distinction between different kinds of nodes also implies that the rules will only merge nodes belonging to the same ontology, and the class names in labels will be replaced by their definition in the corresponding ontology. Finally, blocking distinguishes between nodes which are generated as link successors and nodes created as successors of common roles, since different conditions apply.

When a KB is represented using $\mathcal{E}$-Connections, we can use a set of *optimization techniques* that would not be applicable if the KB had been represented monolithically using a single KB in OWL. All these techniques take advantage of the partitioning of the domains in order to reduce the computational cost:

- **Detection of obvious non-subsumptioms:** Non-subsumptions are hard to detect for tableaux algorithms [16]. Typically, when computing the classification hierarchy of a DL ontology, many subsumption tests that are performed at each node are very likely to fail. These unnecessary tests can be computationally costly and also very repetitive, and hence they affect significantly the performance of a DL reasoner. This problem is usually dealt with using caching, an optimization technique that allows to prove non-subsumptions by using cached results from previous satisfiability tests.

  In an $\mathcal{E}$-Connection, many non-subsumptions become obvious, since a subsumption test $A \sqsubseteq B$ involving two classes A, B belonging to different ontologies in the $\mathcal{E}$-Connection will necessarily fail. However, if these classes were contained in a DL ontology, the satisfiability test could have been actually performed. Hence, the separation of ontologies using $\mathcal{E}$-Connections naturally enhances the effect of caching for avoiding unnecessary subsumption tests.
- **Separation of non-absorbable GCIs:** When general inclusion axioms are present in a knowledge base, a disjunction must be added to the label of each node for each GCI. The resulting increase of non-determinism in the tableau expansion dramatically affects the performance of the DL reasoner. In order to overcome this problem, modern DL reasoners implement an optimization technique called *absorption* [16], which allows to transform many GCIs into primitive definition axioms. However, although absorption typically allows to eliminate most

of the GCIs in a knowledge base, many general axioms may still remain. Non-absorbable GCIs, even in a reduced number, can notably degrade the performance of reasoners. In an $\mathcal{E}$-Connection, non-absorbable GCIs are typically also spread among the different ontologies of the combination. When performing the tableau expansion, a GCI *only* adds a disjunction to the nodes corresponding to the ontology the GCI belongs to, whereas, in the case of a single ontology, the same GCI would add a disjunction in *all* the nodes during the expansion.

- **Separation of ABox individuals:** Typical reasoning services when ABoxes are present are *instantiation* (checking if an individual is an instance of a concept), and *retrieval* (computing all the instances of a certain concept). The presence of a large ABoxes degrades significantly the performance of DL reasoners. When using $\mathcal{E}$-Connections, the ABox axioms and individuals are partitioned among the different ontologies in the combination. This separation is important since it may spare the application of a large number of rules in many satisfiability tests.
- **Optimization of blocking and better use of certain optimization techniques:** Blocking ensures the correct termination of tableau algorithms by limiting the tree expansion depth, which otherwise would become infinite. However, when dealing with logics like $\mathcal{SHIQ}$, the blocking condition is quite sophisticated and blocking may occur late in the tableau expansion. Having a more permissive blocking condition is important to establish blocks at a shallower depth, and can substantially increase performance. Although the restrictions can be made less strict [18], the resulting blocking condition still remains less efficient than subset or equality blocking, which unfortunately are not sound for logics like $\mathcal{SHIQ}$. However, in an $\mathcal{E}$-Connection, blocking is optimized for each of the components, which implies, for example, that subset blocking is still applicable in a component without inverse properties, even if such a component is connected to a $\mathcal{SHIQ}$ component. On the other hand, certain optimization techniques are not valid for certain logics. For example, caching the satisfiability status of nodes within a tableau expansion cannot be performed in logics containing inverse roles. For the same reason as in blocking, optimizations like caching could still be applied to some of the component ontologies in the combination.

11.7 Conclusion and Future Work

In this paper, we have presented $\mathcal{E}$-Connections as a suitable formalism for combining OWL ontologies. We have discussed the applicability and usefulness of $\mathcal{E}$-Connections as a combination technique for many application scenarios.

In order to integrate $\mathcal{E}$-Connections in the Semantic Web, we have extended the syntax and semantics of the OWL-DL recommendation and we have discussed the issues of handling URIs and imports in this new framework. We have provided suitable tool support for browsing and editing $\mathcal{E}$-Connected ontologies in Swoop.

We have shown how to reason with certain families of $\mathcal{E}$-Connections and proved that it is possible to implement our tableau algorithms for $\mathcal{E}$-Connections as an extension of existing DL reasoners, as shown by our implementation in the Pellet system.

Our initial empirical results suggest that reasoning over expressive $\mathcal{E}$-Connections is reasonably efficient and, in practice, it is not harder than reasoning with OWL itself. Finally, we have also identified the limitations of $\mathcal{E}$-Connections as a combination technique, in particular for tasks, such as ontology refinement, that involve connecting ontologies dealing with highly overlapping domains.

In the future, it would be interesting to explore the design and implementation of practical algorithms for combinations of Description Logics with spatial and temporal logics. These combinations are important for many applications, since OWL is not a suitable formalism for representing temporal and spatial information. For instance, in the Wine Ontology example, the ontology about regions could be represented using a qualitative spatial logic, like $\mathbf{S4_u}$, instead of using OWL. In [31] the decidability of such combinations was proved and the development of practical algorithms will provide a strong motivation for bringing these formalisms to the Semantic Web.

References

1. Baader, F., Horrocks, I., Sattler, U.: Description logics as ontology languages for the semantic web. In: Hutter, D., Stephan, W. (eds.) Festschrift in honor of Jörg Siekmann. LNCS (LNAI). Springer, Heidelberg (2003)
2. Baader, F., Lutz, C., Sturm, H., Wolter, F.: Fusions of description logics and abstract description systems. Journal of Artificial Intelligence Research (JAIR) 16, 1–58 (2003)
3. Baader, F., Nutt, W.: Basic description logics. In: Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.F. (eds.) The Description Logic Handbook: Theory, Implementation, and Applications, pp. 43–95. Cambridge University Press, Cambridge (2003)
4. Baader, F., Sattler, U.: An overview of tableau algorithms for description logics. Studia Logica 69, 5–40 (2001)
5. Baader, F., Hladik, J., Lutz, C., Wolter, F.: From tableaux to automata for description logics. Fundamenta Informaticae 57, 1–33 (2003)
6. Bechhofer, S., Lord, P., Volz, R.: Cooking the semantic web with the OWL API. In: Fensel, D., Sycara, K.P., Mylopoulos, J. (eds.) ISWC 2003. LNCS, vol. 2870, pp. 659–675. Springer, Heidelberg (2003)
7. Brickley, D., Guha, R.V.: Resource description framework (RDF) model and syntax specification. W3C Recommendation (1999)
8. Cuenca, B., Parsia, B., Sirin, E.: Working with multiple ontologies on the semantic web. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 620–634. Springer, Heidelberg (2004)
9. Cuenca-Grau, B., Parsia, B., Sirin, E.: Tableau algorithms for e-connections of description logics. Technical report, UMIACS (2004), `http://www.mindswap.org/2004/multipleOnt/papers/EconnTableau.ps`
10. Cuenca Grau, B., Parsia, B., Sirin, E., Kalyanpur, A.: Automatic partitioning of owl ontologies using $mathcalE$-connections. In: Horrocks, I., Sattler, U., Wolter, F. (eds.) Description Logics. CEUR Workshop Proceedings, vol. 147, CEUR-WS.org. (2005)
11. Cuenca Grau, B., Parsia, B., Sirin, E., Kalyanpur, A.: Modularity and web ontologies. In: Doherty, P., Mylopoulos, J., Welty, C.A. (eds.) KR, pp. 198–209. AAAI Press, Menlo Park (2006)

12. Golbeck, J., Fragoso, G., Hartel, F., Hendler, J., Parsia, B., Oberthaler, J.: The national cancer institute's thesaurus and ontology. Journal of Web Semantics 1(1) (December 2003)
13. Hladik, J., Model, J.: Tableau systems for $\mathcal{SHIO}$ and $\mathcal{SHIQ}$. In: Haarslev, V., Möller, R. (eds.) Proceedings of the 2004 International Workshop on Description Logics (DL 2004). CEUR (2004), `ceur-ws.org`
14. Horrocks, I.: Optimising Tableaux Decision Procedures for Description Logics. PhD thesis, University of Manchester (1997)
15. Horrocks, I.: Using an expressive description logic: FaCT or fiction? In: Proc. of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR 1998), pp. 636–647. Morgan Kaufman, San Francisco (1998)
16. Horrocks, I.: Implementation and optimisation techniques. In: Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.F. (eds.) The Description Logic Handbook: Theory, Implementation, and Applications, pp. 306–346. Cambridge University Press, Cambridge (2003)
17. Horrocks, I., Sattler, U.: Ontology reasoning in the $\mathcal{SHOQ}(\mathcal{D})$ description logic. In: Nebel, B. (ed.) Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001), pp. 199–204. Morgan Kaufmann, San Francisco (2001)
18. Horrocks, I., Sattler, U.: Optimised reasoning for $\mathcal{SHIQ}$. In: Proc. of the 15th European Conference on Artificial Intelligence (ECAI 2002) (2002)
19. Horrocks, I., Sattler, U., Tobies, S.: Practical reasoning for very expressive description logics. Logic Journal of the IGPL 8(3), 239–263 (2000)
20. Horrocks, I., Patel-Schneider, P.F., van Harmelen, F.: From $\mathcal{SHIQ}$ and RDF to OWL: The making of a web ontology language. J. of Web Semantics 1(1), 7–26 (2003)
21. Horrocks, I., Sattler, U.: A description logic with transitive and inverse roles and role hierarchies. Journal of Logic and Computation 9(3), 385–410 (1999)
22. Horrocks, I., Sattler, U.: A tableaux decision procedure for SHOIQ. In: Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005) (to appear, 2005)
23. Kalyanpur, A., Parsia, B., Hendler, J.: A tool for working with web ontologies. International Journal on Semantic Web and Information Systems 1(1) (2004)
24. Kutz, O.: $\mathcal{E}$-Connections and Logics of Distance. PhD thesis, University of Liverpool (2004)
25. Kutz, O., Lutz, C., Wolter, F., Zakharyaschev, M.: $\mathcal{E}$-Connections of Abstract Description Systems. Artificial Intelligence 156(1), 1–73 (2004)
26. Patel-Schneider, P.F., Hayes, P., Horrocks, I.: Web ontology language OWL Abstract Syntax and Semantics. W3C Recommendation (2004)
27. Sirin, E., Parsia, B.: Pellet system description. In: Proceedings of the 2004 International Workshop on Description Logics (DL 2004), Whistler, British Columbia, Canada, June 6-8, 2004. CEUR Workshop Proceedings, vol. 104. CEUR-WS.org. (2004)
28. Smith, M.K., Welty, C., McGuiness, D.L.: OWL Web Ontology Language Guide. W3C Recommendation (2004)
29. StuckenSchmidt, H., Klein, M.: Integrity and change in modular ontologies. In: Proceedings of the Eighteenth Joint Conference on Artificial Intelligence (IJCAI 2003). Morgan Kaufmann, San Francisco (2003)
30. StuckenSchmidt, H., Klein, M.: Structure-based partitioning of large concept hierarchies. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 289–303. Springer, Heidelberg (2004)
31. Haarslev, V., Moeller, R.: Racer system description. In: Goré, R.P., Leitsch, A., Nipkow, T. (eds.) IJCAR 2001. LNCS(LNAI), vol. 2083, pp. 701–705. Springer, Heidelberg (2001)

12

Composing Modular Ontologies with Distributed Description Logics

Luciano Serafini and Andrei Tamilin

Center for Information Technology
Fondazione Bruno Kessler
Trento, Italy
{serafini,tamilin}@fbk.eu

Summary. This chapter demonstrates the use of the Distributed Description Logics framework (DDL) and the distributed reasoner DRAGO as formal and practical tools for composing modular ontologies from purely terminological $\mathcal{SHIQ}$ ontology modules. According to DDL vision, a modular ontology can be formally represented by a distributed T-box, comprising a set of separate T-boxes (one for each ontological module), which are pairwise interrelated by “bridge rules” (inter-module connectives allowing to access and import knowledge contained in modules). The chapter gives the semantic explanations of knowledge import via bridge rules as well as presents the distributed tableaux reasoning technique for its computation. Practically, the implementation of the distributed tableaux in DRAGO reasoner and its use for modular ontology composition is described and experimentally evaluated.

12.1 Motivation and Approach

Recent years have witnessed the theoretical birth and practical maturation of the web ontology technology. Significant amount of academic research has been directed on the proposal and later the World Wide Web consortium standardization of representational languages for publishing ontologies on the web, such as OWL [4], proving the appropriateness of Description Logic formalism for being an underpinning theory for performing the formal analysis of ontologies [3, 2, 1, 15, 13], and finally the development of effective reasoning algorithms [12, 18, 19, 25, 26, 16] and practical implementations of automatic inference systems [14, 11, 10] facilitating the automated processing and use of ontologies. Despite the important steps ahead toward strengthening the web ontology technology, the distinguishing feature of the developed representation languages and tools is their monolithic treatment of ontologies and a little support for organizing ontologies in a modular way.

Practical need for ontology modularization and clarification of what a modular ontology is can be perceived from two complementary scenarios. A promising way of dealing with large-scale ontologies is to decompose (partition) them into a collection of smaller, more specific ontologies, which, together with the relations between them, constitute the representation that is semantically equivalent to the original ontology. Conversely, it may be desirable to compose a set of ontologies into a coherent network that can be referred to as a single entity. In both cases, the ultimate ontology can be referred as *modular* – it comprises a set of autonomous ontological modules, which are interrelated by inter-module connectives. The inter-module connectives enable modules to access the knowledge contained in other connected modules and allowing, therefore, the integral use of the whole modular ontology.

In the current proposal of web ontology language (OWL) the support of ontology modularization is represented by the construct owl:imports. The import functionality of OWL, however, has several limitations moderating its modularization capabilities. Theoretically, the semantics of OWL is defined in such a way that all ontologies, imported and importing, share the single global interpretation. As a consequence, imported and importing ontologies cannot describe differently the very same portion of the world, using different perspectives and levels of granularity, without rising a logical contradiction. The practical consequence of the global semantics of OWL results in a global reasoning approach implemented in existing reasoners. The global reasoning approach consists in taking all ontologies participating in the import, combining them together in a unique reasoning space and further reasoning in it. The final shortcoming of owl:imports concerns with its ability to import ontologies only as a whole, while it is often the case in practice that only a certain part of imported ontology is of interest to the importing ontology.

Taking the above disadvantages, we see the following challenging requirements to be satisfied by the practically utilizable ontology modularization approach:

- Supporting *semantic locality* of ontological modules. This requirement addresses the ability of modules to have semantics different from the created modular ontology, being expressed in local languages of different expressive power so that each module can be dealt with a reasoner, specially tuned for it.
- Preserving *loose coupling* and *autonomy* of ontological modules. In other words, modularization technology should be able to keep ontological modules as separate contributors to the desired modular ontology, rather than hardly integrating them in a monolithic entity. This consequently implies that some sort of a distributed reasoning is required to operate over a set of autonomous ontological modules.
- Enabling a *partial reuse* of knowledge from ontological modules. In many practical cases we would like to reuse only part of a certain module, since the other part of the module may be irrelevant to the application domain of the modular ontology or, in the worst case, can even contradict some of its knowledge.
- Preserving *directionality* of knowledge import. This requirement means that the ontology module should not be affected by the modular ontology importing it. For example, if the importing ontology becomes inconsistent the modules it imports should not be automatically rendered as inconsistent too.
- Ensuring a *scalability* of modular ontology technology. Scalability criterion addresses two aspects: representation and reasoning. The scalability of the representation means the ease of large-scale ontology maintenance through its

modularization, while the reasoning scalability concerns with the ability to reason with large-scale ontologies by modularizing them.

In this chapter we demonstrate the use of the Distributed Description Logics framework [5, 20] and the distributed reasoner DRAGO [21] as formal and practical tools for composing modular ontologies. In this work we consider construction of modular ontologies from purely terminological $\mathcal{SHIQ}$ ontologies. Such a restriction is explained by the current limitations of the DDL distributed reasoning technique implemented in DRAGO. According to DDL, a modular ontology is formally encoded into a distributed T-box, comprising a set of T-boxes (one for each ontological module), which are pairwise interrelated by "bridge rules" (inter-module connectives allowing to access and import knowledge contained in modules). The semantics of DDL fits into requirements of semantic locality and directionality, the partial reuse criterion is met due to the use of bridge rules allowing to selectively access the knowledge in the modules, loose coupling and reasoning scalability is met by the use of the distributed reasoning approach of DDL.

The chapter is further organized as follows. In Section 12.2 we introduce the basic definitions of the DDL framework. In Section 12.3 we outline the use of DDL for encoding modular ontologies. In Section 12.4 we present how bridge rules allow to access and import knowledge from modules. The distributed tableaux reasoning technique for automatic computation of knowledge import via bridge rules is presented in Section 12.5. The practical implementation of the distributed reasoner DRAGO and its use for modular ontologies composition is discussed in Section 12.6. The experimental evaluation of the proposed modular framework finalizes the presentation in Section 12.7.

12.2 Introduction to Distributed Description Logics

As introduced by Borgida and Serafini in [5], Distributed Description Logics (DDL) is a framework for representing multiple ontologies *pairwise* interconnected by directional semantic mappings. In this section we overview basic syntactical constructs of DDL, discuss the principle semantic assumptions of the framework, and further describe the supported inference services.

12.2.1 Syntax

Given a set $O = \{O_i\}_{i \in I}$ of ontologies, let $\{\mathcal{DL}_i\}_{i \in I}$ be a set of Description Logics.[1] For each $i \in I$, let us denote a T-box of $\mathcal{DL}_i$ as $\mathcal{T}_i$. Each T-box $\mathcal{T}_i$ contains all the information necessary to define the terminology of an ontology O_i, including not just concept and role definitions, but also general axioms relating descriptions, as well as declarations such as the transitivity of certain roles. Given that setting, the

[1] We assume the reader is familiar with Description Logic and related reasoning systems, as described in [2].

initial set of ontologies O can be formally represented as a corresponding *family of T-boxes* $\mathcal{T} = \{\mathcal{T}_i\}_{i \in I}$.

Since the very same symbol can be used in two ontologies with different meaning, to unambiguously refer to elements of $\mathcal{T}_i$, they are prefixed with the index i of the ontology, e.g., $i : C$, $i : C \sqsubseteq D$, and etc.

To express the links between concepts of different ontologies, DDL introduces a special construct called a bridge rule.

Definition 1 (Bridge rule). *Given a family $\mathcal{T} = \{\mathcal{T}_i\}_{i \in I}$ of terminologies, a* bridge rule *from i to j is an expression of the following two forms:*

$i : X \xrightarrow{\sqsubseteq} j : Y$ – *an* into-bridge rule
$i : X \xrightarrow{\sqsupseteq} j : Y$ – *an* onto-bridge rule

where X, Y are concepts of $\mathcal{T}_i$ and $\mathcal{T}_j$ respectively. The derived equivalence bridge rule $i : X \xrightarrow{\equiv} j : Y$ can be defined as a pair of into- and onto-bridge rules from $i : X$ to $j : Y$. ◇

Bridge rules do not represent semantic relations stated from an external *objective* point of view. Indeed, there is no such point of view on the web. As such, bridge rules from i toward j express directional relations between i and j viewed from the j's *subjective* point of view. Intuitively, the into-bridge rule $i : \mathsf{PhDThesis} \xrightarrow{\sqsubseteq} j : \mathsf{Thesis}$ states that, from the j's point of view, the concept $\mathsf{PhDThesis}$ in i is less general than its local concept Thesis. Conversely, the onto-bridge rule $i : \mathsf{InProceedings} \xrightarrow{\sqsupseteq} j : \mathsf{ConferencePaper}$ expresses the j's point of view that $\mathsf{InProceedings}$ in i is more general than the local concept $\mathsf{ConferencePaper}$ in j.

Note, that since bridge rules reflect a subjective point of view, bridge rules from j to i are not necessarily the inverse of the rules from i to j, and in fact there may be no rules in one or both the directions.

Example 1. Let us consider two simple concept hierarchies, depicted in Figure 12.1, extracted from two ontologies describing the domain of scientific publications.

The following are possible examples of bridge rules from O_1 to O_2:

$$1 : \mathsf{Publication} \xrightarrow{\equiv} 2 : \mathsf{ScientificPaper} \tag{12.1}$$

$$1 : \mathsf{InProceedings} \xrightarrow{\sqsubseteq} 2 : \mathsf{ConferencePaper} \sqcup \mathsf{WorkshopPaper} \tag{12.2}$$

$$1 : \mathsf{InBook} \xrightarrow{\sqsupseteq} 2 : \mathsf{BookArticle} \tag{12.3}$$

◇

Given a family of T-boxes and a possibility to interlace them with bridge rules, we can define the notion of a distributed T-box.

Definition 2 (Distributed T-box). *A distributed T-box $\mathfrak{T} = \langle \mathcal{T}, \mathfrak{B} \rangle$ consists of a collection of T-boxes $\mathcal{T} = \{\mathcal{T}_i\}_{i \in I}$ and a collection of bridge rules $\mathfrak{B} = \{\mathfrak{B}_{ij}\}_{i \neq j \in I}$ between pairs of them.* ◇

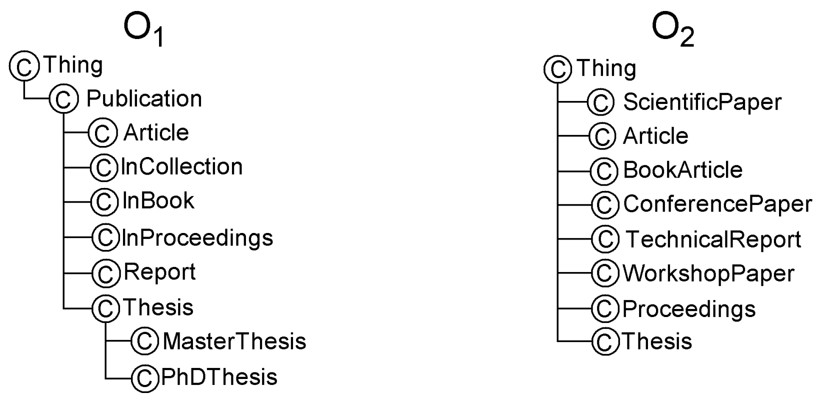


Fig. 12.1. Extracts of concept hierarchies of two ontologies describing domain of publications

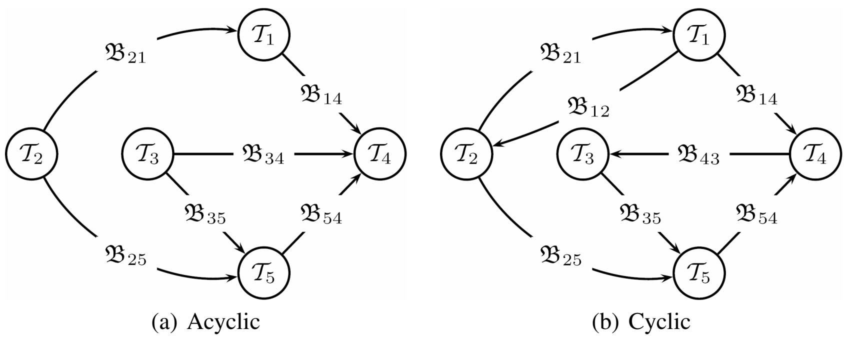


Fig. 12.2. Examples of bridge graphs of distributed T-boxes

In order to characterize the topology of a distributed T-box, DDL utilizes a notion of a bridge graph.

Definition 3 (Bridge graph of a distributed T-box). *Given a distributed T-box $\mathfrak{T} = \langle \mathcal{T}, \mathfrak{B} \rangle$, a* bridge graph $G_{\mathfrak{T}}$ *of a $\mathfrak{T}$ is a directed graph with a set of nodes corresponding to collection of T-boxes $\mathcal{T} = \{\mathcal{T}_i\}_{i \in I}$ and an arc from i to j when the set of bridge rules $\mathfrak{B}_{ij}$ in the family $\mathfrak{B}$ is non-empty.* ◇

As depicted in Figure 12.2(a) and 12.2(b), similarly to the classical graph theory, bridge graphs can be acyclic and cyclic, depending on a family $\mathfrak{B}$ of bridge rules of the distributed T-box.

12.2.2 Semantics

DDL semantics is a customization of the Local Model Semantics for Multi Context Systems [8, 9]. Given a family $\mathcal{T} = \{\mathcal{T}_i\}_{i \in I}$ of ontologies, the fundamental idea of DDL is that all ontologies $\mathcal{T}_i$ are *locally interpreted* on their *local interpretation*

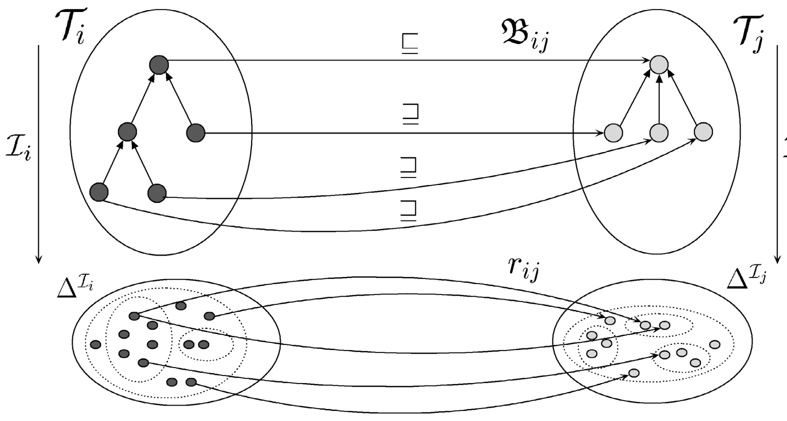


Fig. 12.3. Visualized semantics of DDL

domains. Given that setting, the first component of DDL semantics is a *family of local interpretations* $\{\mathcal{I}_i\}_{i\in I}$, one for each $\mathcal{T}_i$.

In accordance with the definition of interpretation in Description Logic, each interpretation $\mathcal{I}_i$ consists of a non-empty, possibly infinite domain $\Delta^{\mathcal{I}_i}$, and a valuation function $\cdot^{\mathcal{I}_i}$, which maps every concept to a subset of $\Delta^{\mathcal{I}_i}$, every role to a subset of $\Delta^{\mathcal{I}_i} \times \Delta^{\mathcal{I}_i}$. In order to support the directionality of knowledge import via bridge rules, as well as allow to semantically localize inconsistent parts of distributed T-boxes, the DDL framework admits the use of a special hole interpretations as a local interpretation. For the sake of simplicity, we omit the discussion of technical properties of DDL with holes in this work and refer the interested reader to [20].

Since local interpretation domains $\Delta^{\mathcal{I}_i}$ can be heterogeneous, the semantic correspondences between them are modeled using a second component of DDL semantics, a domain relation.

Definition 4 (Domain relation). *A* domain relation r_{ij} *from* $\Delta^{\mathcal{I}_i}$ *to* $\Delta^{\mathcal{I}_j}$ *is a subset of* $\Delta^{\mathcal{I}_i} \times \Delta^{\mathcal{I}_j}$*, such that*

- $r_{ij}(d)$ *denotes* $\{d' \in \Delta^{\mathcal{I}_j} \mid \langle d, d'\rangle \in r_{ij}\}$
- *for any subset* D *of* $\Delta^{\mathcal{I}_i}$, $r_{ij}(D)$ *denotes* $\bigcup_{d\in D} r_{ij}(d)$
- *for any* $R \subseteq \Delta^{\mathcal{I}_i} \times \Delta^{\mathcal{I}_i}$, $r_{ij}(R)$ *denotes* $\bigcup_{\langle d,d'\rangle\in R} r_{ij}(d) \times r_{ij}(d')$ $\diamond$

A domain relation r_{ij} represents a possible way of mapping the elements of $\Delta^{\mathcal{I}_i}$ into the domain $\Delta^{\mathcal{I}_j}$, seen from subjective j-th perspective. For instance, if $\Delta^{\mathcal{I}_1}$ and $\Delta^{\mathcal{I}_2}$ are the representations of cost in US Dollars and in Euro, then r_{12} could be the rate of exchange function, or some other approximation relation.

It should be noted that the domain relation is defined as a generic relation – thus it is *not a function*; it is *not total*; r_{ij} is *not the inverse* of r_{ji}, i.e., $r_{ij} \neq r_{ji}^{-1}$; it is *not compositional*, i.e., in a general setting $(r_{ij} \circ r_{jk}) \neq r_{jk}$.

Figure 12.3 intuitively depicts component elements of DDL semantics. All together these elements form the principle semantic component of DDL – a distributed interpretation.

Definition 5 (Distributed interpretation and satisfiability). *A* distributed interpretation $\mathfrak{I} = \langle\{\mathcal{I}_i\}_{i\in I}, \{r_{ij}\}_{i\neq j\in I}\rangle$ *of a distributed T-box* $\mathfrak{T} = \langle\mathcal{T}, \mathfrak{B}\rangle$ *consists of a family of local interpretations* $\mathcal{I}_i$ *(classical or holes) on local interpretation domains* $\Delta^{\mathcal{I}_i}$ *and a family of domain relations* r_{ij} *between pairs of local domains.*

A distributed interpretation $\mathfrak{I}$ satisfies *a distributed T-box* $\mathfrak{T} = \langle\mathcal{T}, \mathfrak{B}\rangle$*, is called a model of* $\mathfrak{T}$*, if all T-boxes* $\mathcal{T}_i$ *in* $\mathcal{T} = \{\mathcal{T}_i\}_{i\in I}$ *are locally satisfied*

- $\mathcal{I}_i \models \mathcal{T}_i$

and all bridge rules are satisfied

- $r_{ij}(C^{\mathcal{I}_i}) \supseteq D^{\mathcal{I}_j}$ *for all* $i : C \xrightarrow{\sqsupseteq} j : D \in \mathfrak{B}_{ij}$
- $r_{ij}(C^{\mathcal{I}_i}) \subseteq D^{\mathcal{I}_j}$ *for all* $i : C \xrightarrow{\sqsubseteq} j : D \in \mathfrak{B}_{ij}$ ◇

12.2.3 Inference Services

Similarly to reasoning services defined for Description Logic, the fundamental reasoning services of Distributed Description Logics lay in verification of concept satisfiability/subsumption within certain ontology. What makes the difference is that in DDL, besides the ontology itself, the other linked ontologies should be taken into account.

Given a distributed T-box $\mathfrak{T} = \langle\mathcal{T}, \mathfrak{B}\rangle$, the *distributed inference services* can be defined as follows:

Satisfiability: Given a concept C of $\mathcal{T}_i$, C is *satisfiable* in i with respect to $\mathfrak{T}$ if there exist a distributed interpretation $\mathfrak{I}$ of $\mathfrak{T}$ such that $C^{\mathcal{I}_i} \neq \emptyset$.

Subsumption: Given a pair of concepts C and D of $\mathcal{T}_i$, C is *subsumed* by a concept D in i with respect to $\mathfrak{T}$ if for every distributed interpretation $\mathfrak{I}$ of $\mathfrak{T}$ we have that $C^{\mathcal{I}_i} \subseteq D^{\mathcal{I}_i}$. In this case we will write $\mathfrak{T} \models i : C \sqsubseteq D$.

12.3 Modular Ontologies as Distributed T-Boxes in DDL

As it has been outlined in the opening section of this chapter, a modular ontology comprises a set of autonomous ontological modules, which are interrelated by inter-module connectives allowing to access and import knowledge contained in the connected modules. A *modular ontology*, consequently, can be defined as a tuple consisting of a collection of ontological modules plus inter-module connectives between them. Given a module, the representational language of it can be partitioned into the “local language” and “external language”. Local language contains symbols whose meaning is defined locally within a module, whereas the meaning of symbols belonging to external language is defined in the external module and can be accessed following the inter-module links.

Revisiting definitions of the Distributed Description Logics framework given in the previous section, we see DDL as fitting the above vision of modular ontologies. The general idea consists in encoding modules as T-boxes and utilizing bridge rules as means of bringing the meaning of externally defined symbols into a local context of a module. Consequently, a modular ontology can be formally defined as a distributed T-box, in which T-boxes $\mathcal{T}_i$ represent DL formalization of i-th ontological module, and collections of bridge rules $\mathfrak{B}_{ij}$ formalize the inter-module links allowing the module j to access the knowledge contained in the module i.

Given a bridge rule $i : X \xrightarrow{R} j : Y$, with $R \in \{\sqsubseteq, \sqsupseteq, \equiv\}$, the module j views the concept $j : Y$ as externally defined concept, since its meaning is defined in the module i. The use of bridge rules enables the designer selectively reuse knowledge contained in ontological modules without a necessity to import modules as wholes.

12.4 Knowledge Access and Import via Bridge Rules

In this section we shed the light on the capability of bridge rules to transfer knowledge across component ontologies in DDL. This mechanism plays a substantial role in enabling the use of DDL for access and import of knowledge contained in modules, and hence enabling the overall composition of modular ontologies. We start by recalling the knowledge propagation patterns in DDL and then give a characterization to these propagations in terms of operator. Further, we state the result on soundness and completeness of the knowledge propagation in DDL when expressivity of component ontologies is restricted to $\mathcal{SHIQ}$ Description Logic. The material of this section relies on works [5, 20], in which the propagation of knowledge in DDL has been introduced and studied.

12.4.1 Knowledge Propagation Patterns

To describe the capability of bridge rules to propagate knowledge from T-box i (the source) to j (the target) in a distributed T-box $\mathfrak{T} = \langle \mathcal{T}, \mathfrak{B} \rangle$, we apply the following scheme:

$$\frac{(1)\text{ axioms in } i, \quad (2)\text{ bridge rules from } i \text{ to } j}{(3)\text{ axioms in } j}$$

which must be read as: if the axioms in (1) are true in $\mathcal{T}_i$, the bridge rules in (2) are contained in $\mathfrak{B}_{ij}$, then the axioms in (3) must be true in $\mathcal{T}_j$.

The simplest case illustrating the knowledge propagation in DDL is the following:

$$\frac{i : A \sqsubseteq B, \quad i : A \xrightarrow{\sqsupseteq} j : G, \; i : B \xrightarrow{\sqsubseteq} j : H}{j : G \sqsubseteq H} \tag{12.4}$$

Indeed, $G^{\mathcal{I}_j} \subseteq r_{ij}(A^{\mathcal{I}_i}) \subseteq r_{ij}(B^{\mathcal{I}_i}) \subseteq H^{\mathcal{I}_j}$.

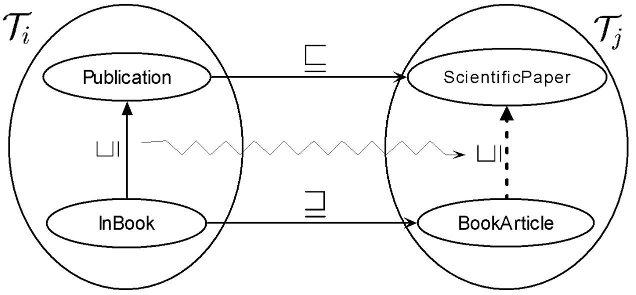


Fig. 12.4. Simple propagation of a subsumption axiom from $\mathcal{T}_i$ to $\mathcal{T}_j$

In other words, a combination of onto- and into-bridge rules allows for propagating subsumption axioms across ontologies. Practically, this means that if an ontology $\mathcal{T}_1$ contains an axiom InBook $\sqsubseteq$ Publication, and an ontology $\mathcal{T}_2$ just defines two concepts ScientificPaper and BookArticle, then bridge rules $1 :$ Publication $\xrightarrow{\sqsubseteq}$ $2 :$ ScientificPaper and $1 :$ InBook $\xrightarrow{\sqsupseteq}$ $2 :$ BookArticle entail in $\mathcal{T}_2$ the axiom that BookArticle is a ScientificPaper. Figure 12.4 illustrates this simple subsumption propagation graphically.

In languages that support disjunction, the simplest propagation rule can be generalized to the propagation of subsumption between a concept and a disjunction of other concepts in the following way:

$$\frac{i : A \sqsubseteq B_1 \sqcup \ldots \sqcup B_n, \quad i : A \xrightarrow{\sqsupseteq} j : G, \; i : B_k \xrightarrow{\sqsubseteq} j : H_k \; (1 \leqslant k \leqslant n)}{j : G \sqsubseteq H_1 \sqcup \ldots \sqcup H_n} \tag{12.5}$$

The important property of the described knowledge propagation is that it is directional, i.e., bridge rules from i to j support knowledge propagation only from i toward j. Moreover, the generalized propagation rule (12.5), as we see further, appears to be the most general form of propagation across $\mathcal{SHIQ}$ ontologies and thus allows capturing correctly and completely terminological propagation in DDL with $\mathcal{SHIQ}$ components.

12.4.2 Soundness and Completeness

To capture the knowledge propagation, DDL proposes to associate with a set $\mathfrak{B}_{ij}$ of bridge rules an operator with the same name, which essentially applies generalized subsumption propagation, rule (12.5), to find a set of subsumption axioms in j-th T-box which are the result of propagation, via bridge rules, of subsumption axioms in i-th T-box.

Definition 6 (Bridge operator). *Given a set of bridge rules $\mathfrak{B}_{12}$ from $\mathcal{T}_1$ to $\mathcal{T}_2$, an operator $\mathfrak{B}_{12}(\cdot)$, taking as input $\mathcal{T}_1$ and producing a set of axioms for $\mathcal{T}_2$, is defined as follows:*

$$\mathfrak{B}_{12}(\mathcal{T}_1) = \left\{ G \sqsubseteq \bigsqcup_{k=1}^{n} H_k \;\middle|\; \begin{array}{l} \mathcal{T}_1 \models A \sqsubseteq \bigsqcup_{k=1}^{n} B_k \\ 1 : A \xrightarrow{\sqsupseteq} 2 : G \;\; \in \mathfrak{B}_{12} \\ 1 : B_k \xrightarrow{\sqsubseteq} 2 : H_k \in \mathfrak{B}_{12} \\ 1 \leqslant k \leqslant n \, , \, n \geqslant 0^2 \end{array} \right\} \quad \diamond$$

The bridge operator contains essentially *all* the inferences that one can get by combining *into*- and *onto*-bridge rules, which can be expressed by the following theorem.

Theorem 1. *Let $\mathfrak{T}_{12} = \langle \{\mathcal{T}_1, \mathcal{T}_2\}, \{\mathfrak{B}_{12}\} \rangle$ be a distributed T-box with two component $\mathcal{SHIQ}$ T-boxes, $\mathcal{T}_1$ and $\mathcal{T}_2$, and a unidirectional set of bridge rules $\mathfrak{B}_{12}$ from $\mathcal{T}_1$ to $\mathcal{T}_2$, then for every pair X and Y of $\mathcal{SHIQ}$ concepts or roles of $\mathcal{T}_2$*

$$\mathfrak{T}_{12} \models 2 : X \sqsubseteq Y \quad \Longleftrightarrow \quad \mathcal{T}_2 \cup \mathfrak{B}_{12}(\mathcal{T}_1) \models X \sqsubseteq Y$$

Some remarkable outcomes:

Upper bound and complexity: If the mapping from 1 to 2 is finite and contains m into-bridge rules and n onto-bridge rules, then the bridge operator $\mathfrak{B}_{12}$ applied to a distributed T-box generates at most $n * 2^m$ subsumption statements. Since the propagation of statements needs checking subsumption in the source T-box, which is EXPTIME complete, we have that computing subsumption in a distributed setting is EXPTIME complete in the dimension of the source T-box plus links.

Vanilla implementation: The above theorem supports a vanilla implementation of *forward chaining* inference engine for DDL. The implementation consists of three steps: computation of propagation operator $\mathfrak{B}_{12}(\mathcal{T}_1)$, construction of extended version of T-box $\mathcal{T}_2$ as $\mathcal{T}_2 \cup \mathfrak{B}_{12}(\mathcal{T}_1)$, and finally applying to this T-box one of existing DL reasoners, such as FaCT++ [27], Racer [11], or Pellet [22].

This approach to reasoning has a strong advantage of reuse of existing highly optimized DL reasoners, however it can be very costly for situations when semantic mappings are changed dynamically or when the required number of reasoning questions to be verified is relatively small. In the next section, we overview an alternative, *backward chaining* reasoning approach, which does "lazy", or on demand, computation of propagated axioms.

As it has been shown in [20], the results of Theorem 1 can be generalized to arbitrary distributed T-boxes $\mathfrak{T} = \langle \mathcal{T}, \mathfrak{B} \rangle$, containing more than two components. According to the generalization, the operator $\mathfrak{B}$ for a family of T-boxes can be defined as follows:

$$\mathfrak{B}(\{\mathcal{T}_i\}_{i \in I}) = \left\{ \mathcal{T}_i \cup \bigcup_{j \neq i} \mathfrak{B}_{ji}(\mathcal{T}_j) \right\}_{i \in I}$$

[2] In case when $n = 0$ we will denote $\bigsqcup_{k=1}^{0} D_k$ as $\bot$.

The remarkable thing is that if I is finite and each $\mathfrak{B}_{ij}$ is finite, then there is a positive integer b such that for every family $\mathcal{T} = \{\mathcal{T}_i\}_{i \in I}$ of T-boxes $\mathfrak{B}^b(\mathcal{T}) = \mathfrak{B}^{b+1}(\mathcal{T})$. If further we define operator $\mathfrak{B}^*(\mathcal{T})$ as $\mathfrak{B}^b(\mathcal{T})$ taking as b the first positive integer witnessing the fixed point, then the following generalized result can be stated.

Theorem 2 (Soundness and Completeness). *For every distributed T-box* $\mathfrak{T} = \langle \mathcal{T}, \mathfrak{B} \rangle$ *and for every pair* X *and* Y *of* $\mathcal{SHIQ}$ *concepts or roles of* $\mathcal{T}_i$

$$\mathfrak{T} \models i : X \sqsubseteq Y \Longleftrightarrow \mathfrak{B}^*(\mathcal{T})_i \models X \sqsubseteq Y$$

where $\mathfrak{B}^*(\mathcal{T})_i$ *is the* i*-th T-box in* $\mathfrak{B}^*(\mathcal{T})$.

12.5 Distributed Reasoning Technique

In this section, we use theoretical results of the previous sections in order to define a tableau-based decision procedure that is capable of checking subsumption (satisfiability) in DDL, i.e., if $\mathfrak{T} \models i : X \sqsubseteq Y$ ($\mathfrak{T} \models i : X$) for some concepts X, Y. Note that similarly to Description Logic, checking concept satisfiability can be reduced to checking subsumption and vice versa [2]. For example, given concepts C and D, $i : C \sqsubseteq D \Longleftrightarrow i : C \sqcap \neg D$ is unsatisfiable.

12.5.1 Intuition

The proposal, which we would like to pursue further in this section, consists in building a decision procedure for a set $\mathcal{T} = \{\mathcal{T}_i\}_{i \in I}$ of ontologies which is distributed among them. More precisely, such a reasoning procedure is constructed as a distributed combination of *local decision procedures*, which decide subsumption/satisfiability with respect to ontologies $\mathcal{T}_i$. Prior to formal introduction of the distributed reasoning procedure, let us consider a simple example revealing the intuition behind.

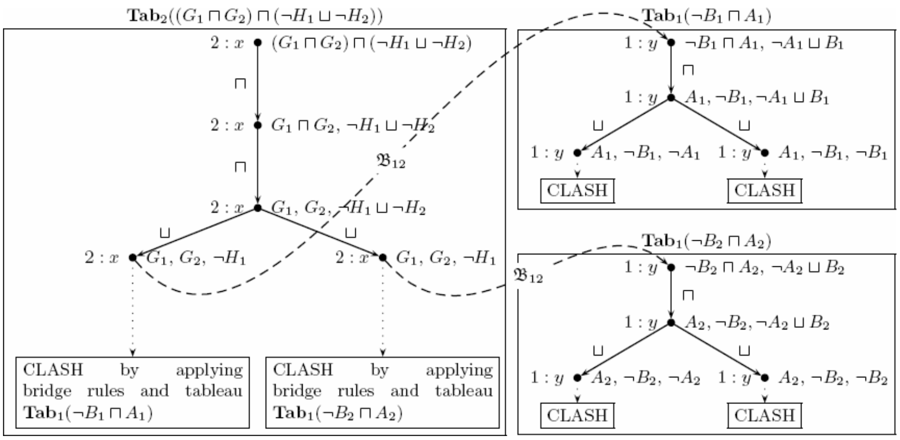
Example 2. Suppose that $\mathfrak{T}_{12}$ is a two-component distributed T-box with a set of unidirectional bridge rules between components, i.e., $\mathfrak{T}_{12} = \langle \{\mathcal{T}_1, \mathcal{T}_2\}, \{\mathfrak{B}_{12}\} \rangle$. Furthermore, suppose that $\mathcal{T}_1$ contains axioms $A_1 \sqsubseteq B_1$ and $A_2 \sqsubseteq B_2$, whereas $\mathcal{T}_2$ does not contain any axiom. Given the set of bridge rules

$$1 : B_1 \xrightarrow{\sqsubseteq} 2 : H_1 \qquad 1 : B_2 \xrightarrow{\sqsubseteq} 2 : H_2 \tag{12.6}$$

$$1 : A_1 \xrightarrow{\sqsupseteq} 2 : G_1 \qquad 1 : A_2 \xrightarrow{\sqsupseteq} 2 : G_2 \tag{12.7}$$

let us show that $\mathfrak{T}_{12} \models 2 : G_1 \sqcap G_2 \sqsubseteq H_1 \sqcap H_2$, i.e., that for any distributed interpretation $\mathfrak{I} = \langle \{\mathcal{I}_1, \mathcal{I}_2\}, \{r_{12}\} \rangle$, $(G_1 \sqcap G_2)^{\mathcal{I}_2} \subseteq (H_1 \sqcap H_2)^{\mathcal{I}_2}$.

1. Suppose that by contradiction there exists $x \in \Delta_2$ such that $x \in (G_1 \sqcap G_2)^{\mathcal{I}_2}$ and $x \notin (H_1 \sqcap H_2)^{\mathcal{I}_2}$.
2. Then $x \in G_1^{\mathcal{I}_2}$, $x \in G_2^{\mathcal{I}_2}$ and either $x \notin H_1^{\mathcal{I}_2}$ or $x \notin H_2^{\mathcal{I}_2}$.



$\mathbf{Tab}_2((G_1 \sqcap G_2) \sqcap (\neg H_1 \sqcup \neg H_2))$
$2 : x$ $(G_1 \sqcap G_2) \sqcap (\neg H_1 \sqcup \neg H_2)$
$2 : x$ $G_1 \sqcap G_2, \neg H_1 \sqcup \neg H_2$
$2 : x$ $G_1, G_2, \neg H_1 \sqcup \neg H_2$
$2 : x$ $G_1, G_2, \neg H_1$
$2 : x$ $G_1, G_2, \neg H_1$
CLASH by applying bridge rules and tableau $\mathbf{Tab}_1(\neg B_1 \sqcap A_1)$
CLASH by applying bridge rules and tableau $\mathbf{Tab}_1(\neg B_2 \sqcap A_2)$
$\mathfrak{B}_{12}$
$\mathbf{Tab}_1(\neg B_1 \sqcap A_1)$
$1 : y$ $\neg B_1 \sqcap A_1, \neg A_1 \sqcup B_1$
$1 : y$ $A_1, \neg B_1, \neg A_1 \sqcup B_1$
$1 : y$ $A_1, \neg B_1, \neg A_1$
$1 : y$ $A_1, \neg B_1, \neg B_1$
CLASH
CLASH
$\mathbf{Tab}_1(\neg B_2 \sqcap A_2)$
$\mathfrak{B}_{12}$
$1 : y$ $\neg B_2 \sqcap A_2, \neg A_2 \sqcup B_2$
$1 : y$ $A_2, \neg B_2, \neg A_2 \sqcup B_2$
$1 : y$ $A_2, \neg B_2, \neg A_2$
$1 : y$ $A_2, \neg B_2, \neg B_2$
CLASH
CLASH

Fig. 12.5. Illustration of the distributed tableaux intuition for Example 2

3. Let us consider the case where $x \notin H_1^{\mathcal{I}_2}$. From the fact that $x \in G_1^{\mathcal{I}_2}$, by the bridge rule (12.7), there exists $y \in \Delta^{\mathcal{I}_1}$ with $\langle y, x \rangle \in r_{12}$ such that $y \in A_1^{\mathcal{I}_1}$.
4. From the fact that $x \notin H_1^{\mathcal{I}_2}$, by bridge rule (12.6), we can infer that for all $y \in \Delta^{\mathcal{I}_1}$ if $\langle y, x \rangle \in r_{12}$ then $y \notin B_1^{\mathcal{I}_1}$.
5. But, since $A \sqsubseteq B \in \mathcal{T}_1$, then $y \in B_1^{\mathcal{I}_1}$ and this is a contradiction.
6. The case where $x \notin H_2^{\mathcal{I}_2}$ is analogous. $\diamond$

As shown in Figure 12.5, the reasoning above can be seen as a combination of a tableau in $\mathcal{T}_2$ with a tableau in $\mathcal{T}_1$. Note that for the sake of simplicity the depicted tableaux are given in the standard logic tableau notation so that a tableau contains all possible branches created during the expansions. This is slightly different from the tableau notation in Description Logic in which the distinct branch is supposed to be a tableau. In the following section we clarify the formal definitions of tableau technique for Description Logic.

12.5.2 Distributed $\mathcal{SHIQ}$ Tableaux Algorithm

In this section we present a distributed tableaux algorithm for reasoning in DDL. The main design idea consists in constructing a network of standard DL tableaux, one for each of ontologies in DDL, which communicate via mappings in a backward fashion.

Since we restricted the expressivity of ontologies participating in DDL to $\mathcal{SHIQ}$ DL, we will consider in the following that ontologies $\mathcal{T}_i$ are attached with $\mathcal{SHIQ}$-tableau reasoning procedures [17]. For sake of clarity, we first consider the case of a distributed T-box $\mathfrak{T}_{12} = \langle \{\mathcal{T}_1, \mathcal{T}_2\}, \{\mathfrak{B}_{12}\} \rangle$ composed from two components and unidirectional bridge rules between them. Further we generalize the results to distributed T-boxes with arbitrary bridge graphs $G_{\mathfrak{T}=\langle \mathcal{T}, \mathfrak{B} \rangle}$.

We need the usual notion of axiom internalization [17]: given a T-box $\mathcal{T}_i$, the concept $C_{\mathcal{T}_i}$ is defined as $C_{\mathcal{T}_i} = \bigsqcap_{E \sqsubseteq D \in \mathcal{T}_i} \neg E \sqcup D$; also, the role hierarchy $R_{\mathcal{T}_i}$ contains the role axioms of $\mathcal{T}_i$, plus additional axioms $P \sqsubseteq U$, for each role P of $\mathcal{T}_i$, with U some fresh role.

The algorithm for testing j-satisfiability of a concept expression X (i.e., checking $\mathfrak{T} = \langle \mathcal{T}, \mathfrak{B} \rangle \not\models j : X \sqsubseteq \bot$) builds, as usual, a finite representation of a distributed interpretation $\mathfrak{I}$, by running local *autonomous* $\mathcal{SHIQ}$ tableaux procedures to find each local interpretation $\mathcal{I}_i$ of $\mathfrak{I}$.

Definition 7 (DTab$_j$). *For each $j \in I$, let us define a function* **DTab**$_j$ *which takes as an input a concept X and further tries to build a representation of $\mathcal{I}_j$ with $X^{\mathcal{I}_j} \neq \emptyset$ (called a* completion tree *[17]) for the concept $X \sqcap C_{\mathcal{T}_j} \sqcap \forall U.C_{\mathcal{T}_j}$, by applying $\mathcal{SHIQ}$tableau expansion rules, w.r.t. the role hierarchy $R_{\mathcal{T}_j}$, plus the following additional "bridge" expansion rule*

$$\mathfrak{B}_{ij}\text{-}\mathit{rule:}\quad \boxed{\begin{array}{ll} \mathit{if}\ 1. & G \in \mathcal{L}(x),\ \mathit{such\ that}\ i : A \xrightarrow{\sqsupseteq} j : G \in \mathfrak{B}_{ij}, \\ & \mathbf{H} \subseteq \{H_k \mid i : B_k \xrightarrow{\sqsubseteq} j : H_k \in \mathfrak{B}_{ij}\}, \\ & \mathbf{B} = \{B_k \mid H_k \in \mathbf{H},\ i : B_k \xrightarrow{\sqsubseteq} j : H_k \in \mathfrak{B}_{ij}\}, \\ 2. & \mathbf{DTab}_i(A \sqcap \neg \bigsqcup \mathbf{B}) = \mathit{Unsatisfiable\ for} \bigsqcup \mathbf{H} \notin \mathcal{L}(x), \\ & \\ \mathit{then} & \mathcal{L}(x) \longrightarrow \mathcal{L}(x) \cup \{\bigsqcup \mathbf{H}\} \end{array}}$$

◇

The idea behind the **DTab**$_j$ procedures is inspired by Definition 12.6 of the bridge operator $\mathfrak{B}_{ij}(\cdot)$. Whenever **DTab**$_j$ encounters a node x that contains a label G which is a consequence of an onto-bridge rule, then if $G \sqsubseteq \bigsqcup \mathbf{H}$ is entailed by the bridge rules, the label $\bigsqcup \mathbf{H}$, is added to x. To determine if $G \sqsubseteq \bigsqcup \mathbf{H}$ is entailed by bridge rules $\mathfrak{B}_{ij}$, **DTab**$_j$ invokes **DTab**$_i$ on the satisfiability of the concept $A \sqcap \neg \bigsqcup \mathbf{B}$. In its turn, **DTab**$_i$ will build (independently from **DTab**$_j$) an interpretation $\mathcal{I}_i$.

Example 3. To illustrate invocation and execution of **DTab**$_j$ functions, let us consider the setting introduced in Example 1. Figure 12.6 depicts the trace of satisfiability test of 2 : BookArticle $\sqcap \neg$ScientificPaper concept. Notice that at step (2) **DTab**$_2$ applies $\mathfrak{B}_{ij}$-rule and invokes **DTab**$_1$ in accordance with the bridge rules (12.1) and (12.3). ◇

Theorem 3 (Termination, Soundness, Completeness). *Given $\mathcal{SHIQ}$ DL T-boxes $\mathcal{T}_1$ and $\mathcal{T}_2$, let $\mathfrak{T}_{12} = \langle \{\mathcal{T}_1, \mathcal{T}_2\}, \mathfrak{B}_{12} \rangle$ be a distributed T-box. Then, given a $\mathcal{SHIQ}$ concept X*

1. *a distributed procedure* **DTab**$_2(X)$ *terminates, and*
2. *X is satisfiable in $\mathcal{T}_2$ with respect to $\mathfrak{T}_{12}$ if and only if* **DTab**$_2(X)$ *yields a complete and clash-free completion tree.*

DTab$_2$(BookArticle $\sqcap$ $\neg$ScientificPaper)

Step	Tree	Labeling	Expansion rule
(1)	x	$\mathcal{L}(x) = \{$BookArticle $\sqcap$ $\neg$ScientificPaper$\}$	$\sqcap$-rule
(2)*	x	$\mathcal{L}(x) = \{$BookArticle, $\neg$ScientificPaper$\}$	$\mathfrak{B}_{ij}$-rule
(3)	x	$\mathcal{L}(x) = \{$BookArticle, $\neg$ScientificPaper, ScientificPaper$\}$	
(4)	x	$\mathcal{L}(x) = \{\text{Clash}\}$	

DTab$_1$(InBook $\sqcap$ $\neg$Publication)

Step	Tree	Labeling	Expansion rule
(1)	x	$\mathcal{L}(x) = \{$InBook $\sqcap$ $\neg$Publication$\}$	
		$\vdots$	$\mathcal{SHIQ}$-rules
(n)	x	$\mathcal{L}(x) = \{\text{Clash}\}$	

Fig. 12.6. Illustration of satisfiability test of 2 : BookArticle $\sqcap$ $\neg$ScientificPaper

Some remarkable observations and generalizations of the above theorem:

Application to parallelization: The construction of the distributed interpretation proposed in the distributed tableaux algorithm enjoys parallelization; each local tableau procedure **DTab**$_j$ can run independently from the others, without checking for standard $\mathcal{SHIQ}$-blocking conditions with nodes generated by the other tableaux.

Generalization to cycles: The second observation concerns with the possibility of generalization of the distributed tableaux algorithm to the case of distributed T-boxes with *arbitrary* bridge graphs. To prevent the infinite looping due to the possible cycles in the bridge graph and guarantee the termination of the distributed procedure, we can apply the standard cycle resolution strategy used in distributed query answering systems. Namely, the initial satisfiability request to a certain **DTab** procedure is marked with a unique identifier, say id, which is later on used for marking all of the requests generated by bridge expansion rules to other **DTab**s. If during traversing the bridge graph the request generates itself, then it is required to be blocked, i.e., **DTab** returns value "*Satisfiable*" and thus bridge expansion rules are not applied.

Application to caching: A number of researchers have considered the idea of caching locally the necessary information from the imported ontology $\mathcal{T}_{other}$, since this is assumed to be both more efficient (there is no need to interrupt local reasoning, while waiting for answers from the other ontology), and more perspicuous from the point of view of the local user: in order to understand an imported concept D, it is not necessary to understand all of $\mathcal{T}_{other}$, only the locally cached part, which is presumed to be much smaller. Such an approach to the task of reasoning with modular ontologies is exploited in [23]. Theorem 2 of the previous section indicates that it is possible to finitely pre-compile in a sound and complete manner the subsumption information imported into an ontology $\mathcal{T}_j$ by bridge rules in a distributed T-box $\mathfrak{T}$: compute and store it.

12.6 Distributed Reasoning with DRAGO

In this section we overview the design and implementation principles that lay in the base of DRAGO (Distributed Reasoning Architecture for a Galaxy of Ontologies), the system for reasoning with multiple ontologies interconnected by pairwise links.[3] The material of the section relies on the original proposal of DRAGO by Serafini and Tamilin in [21].

12.6.1 General Vision

As depicted in Figure 12.7, DRAGO envisages a setting when multiple ontologies together with semantic mappings between them are distributed amongst a peer-to-peer network of DRAGO Reasoning Peers, or shortly DRP.

Each peer DRP_p hosts an ontology O_p together with a set of semantic mappings $\{M_{pi}\}_{i \neq p}$ incoming to O_p from other ontologies O_i. Furthermore, the DRP hosts a standard tableau-based DL reasoner R_p (FaCT [14], RACER [11], Pellet [10], etc.) which is supposed to be optimally tuned for the local language of O_p. The tableau algorithm of each local reasoner R_p is modified by enriching it with additional expansion rules in accordance with the distributed reasoning algorithm presented in the previous section.

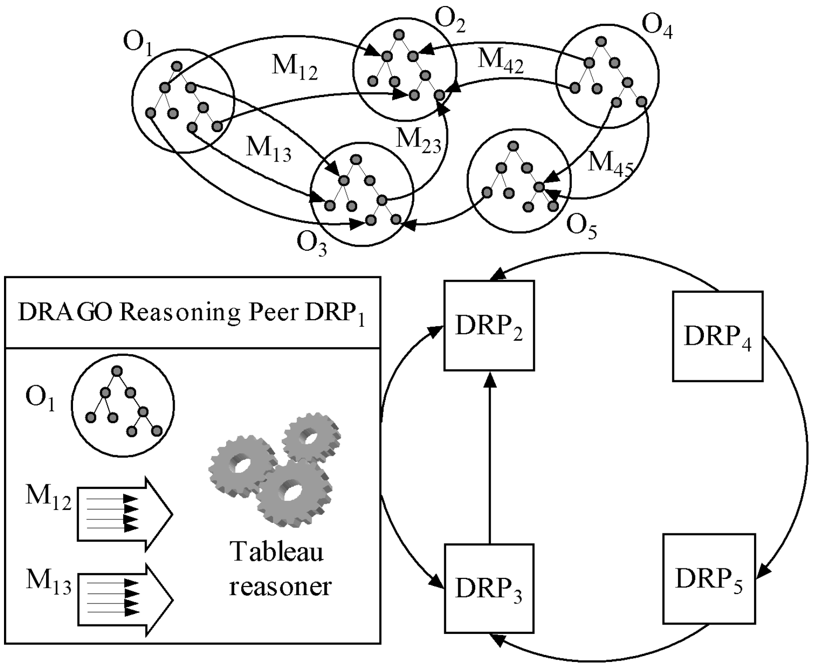


Fig. 12.7. Peer-to-peer reasoning vision of DRAGO

[3] The DRAGO system is available for download from `http://drago.fbk.eu`

The modifications, performed to local reasoners, allow hosting DRPs to invoke their reasoners in two inference modes: local and distributed. The local reasoning mode consists in executing the standard DL tableau functionality of a reasoner. Conversely, the distributed reasoning mode applies the bridge expansion rules in order to take into account semantic mappings and additional knowledge induced by them. As such, the distributed reasoning mode makes reasoning peers communicate via mappings.

Services provided by each reasoning peer can be partitioned into three categories:

- Registration service allows attaching ontology and semantic mappings to a reasoning peer.
- Reasoning services offer access to local and distributed peers' inference functionality over the hosting ontology.
- Communication service is responsible for propagating reasoning requests to other, foreign, reasoning peers.

The registration and reasoning services are public services, thus they can be invoked by users or applications working with reasoning peers. While the communication service is private – it is invoked only by a peer itself in order to establish communication channels with other peers when this is required by the distributed tableaux reasoning algorithm.

12.6.2 Architecture

The DRAGO Reasoning Peer is a principle building block of the DRAGO system. Each peer, being started, gets a reference address, allowing other peers or other applications to request its functionality.

Figure 12.8 depicts the unfolded view of DRP showing its major functional components. Let us stepwise describe the role and the details of each of the components.

Registration Service. In order to host an ontology with a collection of attached semantic mappings, a user or application invokes the Registration Service of a DRP and sends to it the following registration information:

- URL of the local ontology,
- collection of URLs for semantic mappings attached to the local ontology,
- for each of foreign ontologies participating in the attached semantic mapping, the reference address of DRPs hosting them should be specified; doing that we acquaintance the current DRP with the other peers so that the future distributed reasoning is possible.

The registration procedure involves the following components of DRP. The *Parser* component translates the peer ontology and mappings to the object representation of DL knowledge base. The ontology parser is tailored on ontology languages (e.g., OWL/RDF [4]). The result of the ontology parsing consists in extraction of T-box. The mapping parser is tailored on semantic mapping languages (e.g., C-OWL [6]). The result of its application is a set of extracted semantic correspondences (M-box).

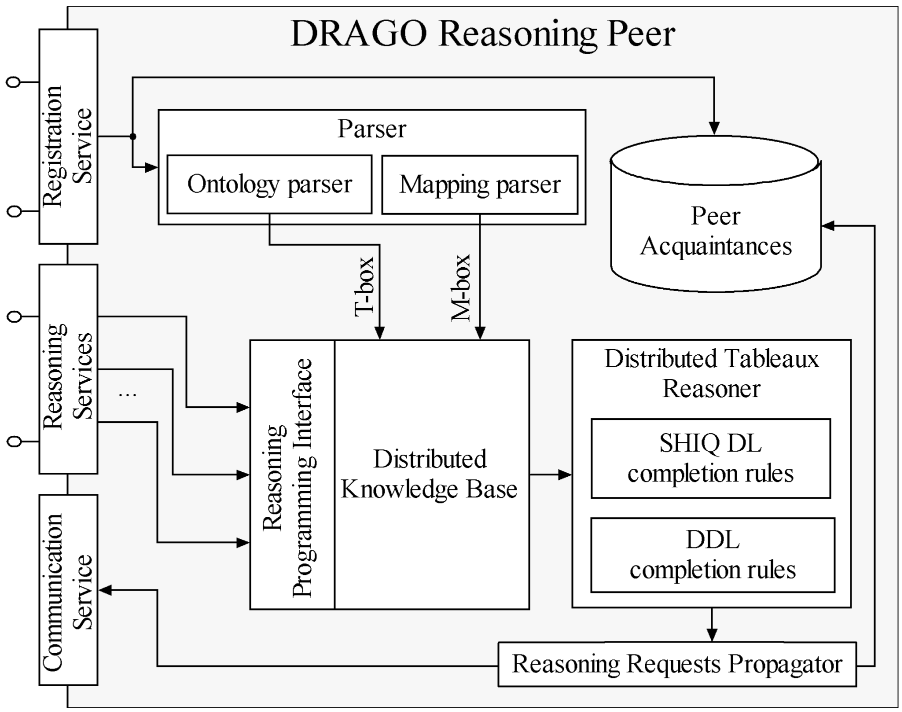


Fig. 12.8. Architecture of the DRAGO Reasoning Peer

All steps together form the tuple ⟨T-box, M-box⟩ that constitutes a *Distributed Knowledge Base* of the DRP.

The *Acquaintances* storage accumulates the addresses of other DRPs whose ontologies participate in the attached mappings. As we see further, the storage is consulted whenever some reasoning sub task is generated by the DRP reasoner and is needed to be propagated to the responsible peer.

Reasoning Services. After the completion of the registration phase the DRP is ready to be started and serve reasoning request for the hosting ontology. Once the reasoning request is caught, the peer invokes corresponding method of the *Reasoning Programming Interface* layer asking through it the Distributed Knowledge Base component to reformulate the reasoning query for evaluating it by the tableau reasoning algorithm. Practically, the Distributed Knowledge Base applies the reduction of DL inference tasks to the one inference task implemented in the reasoner, which is typically the consistency check. The reformulated query is further delivered to the *Reasoner* and processed.

The *Reasoner* is the central component of the DRP. As we already pointed out in the vision section, the DRP reasoner is implemented on top of standard DL reasoner whose standard tableau algorithm is modified by adding additional completion rules allowing for the distributed reasoning. This modification allows the reasoner to be used in two modes: local and distributed. Local reasoning mode simply ignores mappings and applies the standard reasoning algorithm. The distributed mode uses the mappings contained in the M-box and employs the tableau algorithm enriched with additional bridge expansion rule. In accordance with the distributed reasoning algorithm, the tableau completion in the distributed mode may require verification, induced by the mappings, of certain sub reasoning queries over other ontologies. To find the DRP hosting the desired ontology and propagate the query to it, the current peer first employs the *Reasoner Request Propagator* which consults *Acquaintances* base to find out the address of the DRP hosting the ontology to be asked, and further refers to the *Communication Service* for dispatching the query.

Communication Service. Communication service represents a subordinate service which is invoked only internally by the peer. Its main task consists in dispatching the reasoning requests to "foreign" DRPs and delivering back the responses. Technically the cross DRP communication in DRAGO is organized in a message-passing fashion adopting a simple textual communication protocol.

12.6.3 Implementation

Input Languages. Practically, DRAGO works with ontologies represented in OWL [4] and semantic mappings encoded in C-OWL [6, 7]. Due to the restrictions of the distributed reasoning algorithm to $\mathcal{SHIQ}$ ontologies, DRAGO prohibits the use of nominal-related constructs in OWL, such as owl:oneOf and owl:hasValue.

Distributed Tableaux Reasoner. In order to build the Reasoner component of the DRP, any of the standard tableau-based reasoner should be extended with the additional bridge expansion rule. Such a modification of the completion strategy couldn't be organized using the reasoner as a black-box since none of the currently existing reasoners provide an access to the constructed completion graph. That is why the selection of open-source DL reasoner for implementing DRAGO was crucial.

For the modifications, we have selected an open-source OWL-DL Reasoner Pellet written in java programming language [10, 22]. Pellet is based on the tableaux algorithms developed for very expressive Description Logics [16] and covers completely the DL fragment of OWL language. In order to support future extensions, the internals of Pellet tableaux reasoner have been built on an extensible architecture. In particular, the completion algorithm inside the tableau reasoner has been designed so that different Completion Strategies can be plugged in which gave us a perfect opportunity to add the developed distributed completion strategy.

12.6.4 Using DRAGO for Modular Ontology Reasoning

The implementation of modular ontology reasoning with DRAGO becomes a straightforward task consisting of the following steps. For each module i the DRAGO Reasoning Peer should be instantiated. The peer should be further attached with an ontology i and a set of possible inter-module links $\mathfrak{B}_{ki}$ incoming to the module i from other modules k. After that step all modules' peers should be started which forms the DRAGO network providing the reasoning infrastructure for the given modular ontology.

12.7 Experimental Evaluation

We have conducted a number of experiments for evaluating the reasoning with modular ontologies. The goal of the evaluation was to address the requirement of scalability of modular ontology reasoning. Given a modular ontology, we measured the performance of the DDL distributed reasoning algorithm implemented in DRAGO against the performance of the standard DL reasoning algorithm employing the encoding of DDL into standard DL as proposed in [5].

12.7.1 Experimental Setting

In order to perform the tests, we have selected several real world ontologies of different size and complexity. Some of the ontologies have been borrowed from the OntoFarm project [24], the others has been downloaded from on-line Protégé ontology library.[4] Table 12.1 describes the ontologies we used, specifying their size and expressive power in terms of the underlying logic.

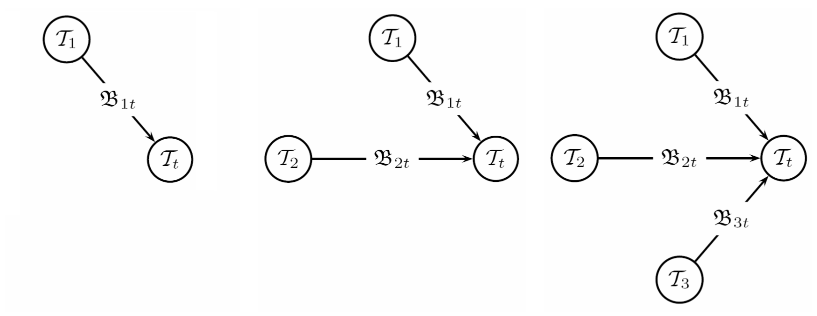
Table 12.1. Modular ontology reasoning: ontologies test set

Ontology	DL Expressivity	Number of classes	Number of properties
CMT	$\mathcal{ALCIF}(\mathcal{D})$	30	59
TRAVEL	$\mathcal{ALCIF}(\mathcal{D})$	51	71
EKAW	$\mathcal{SHIN}$	73	33
KA	$\mathcal{AL}(\mathcal{D})$	96	60
OPENGALEN	$\mathcal{SHIF}$	175	73

Modular ontologies to be tested have been generated synthetically using ontologies in Table 12.1 as modules. We evaluated three topologies of modular ontologies, namely when there exists a target module $\mathcal{T}_t$ which is connected via bridge rules to one source module $\mathcal{T}_1$, connected to two source modules $\mathcal{T}_1$ and $\mathcal{T}_2$, and finally connected to three source modules $\mathcal{T}_1$, $\mathcal{T}_2$, and $\mathcal{T}_3$ (see Figure 12.9 for visual representations of the evaluated topologies).

To synthesize a set of experimental modular ontologies, we have used as source and target modules the very same ontology from the Table 12.1. Bridge rules between modules have been generated randomly varying their amount. The idea was to investigate how the percent of involvement of concepts in target module in bridge rules affects the reasoning in the target module. In particular, we have evaluated the 20,

[4] `http://protege.stanford.edu/plugins/owl/owl-library/`



(a) 1 source module (b) 2 source modules (c) 3 source modules

Fig. 12.9. Topologies of test modular ontologies

40, 60, 80 and 100% coverage of target ontology by bridge rules. In case of several source modules, the bridge rules have been considered equally distributed between sources.

Given a synthesized modular ontology, we further employed the following evaluation scheme:

- Initialize the DRAGO network with the modular ontology and measure the CPU time spent by the reasoner to load modules ($dLoad$ variable). The time spent by the reasoner is the time the DL tableau algorithm requires for standard knowledge base preprocessing.
- Submit 50 random distributed satisfiability tests to be verified in target module $\mathcal{T}_t$, and measure the average time spent by distributed reasoning algorithm of DRAGO for verification of a single satisfiability request ($dSat$ variable).
- Create a global ontology equivalent to the modular ontology using the encoding described in [5]. Load it to the Pellet DL reasoner and measure the time spent for preprocessing this global ontology ($gLoad$ variable).
- Submit the same set of 50 satisfiability tests to the global reasoner and measure the average time spent for deciding satisfiability of a single satisfiability request ($gSat$ variable).

12.7.2 Results

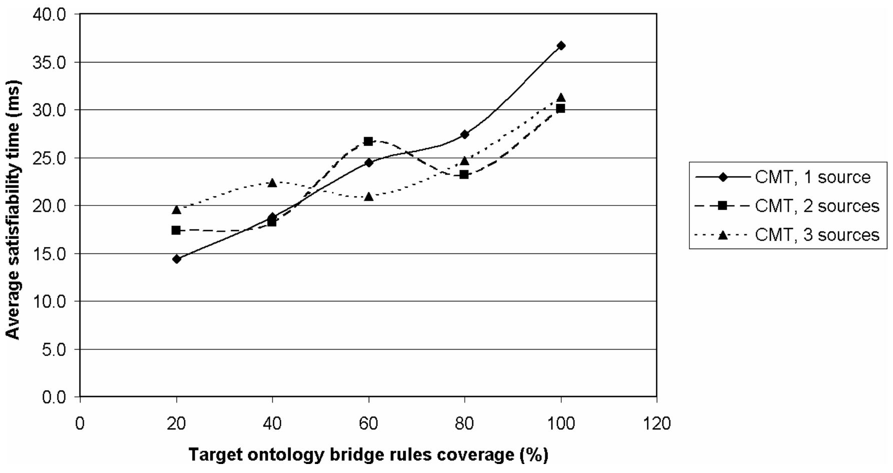
All tests and measurements reported in this section have been carried out on Intel Pentium M processor 2.00GHz with 1.00 GB of RAM running Microsoft Windows XP Professional.

The results of the conducted evaluation are summarized in Table 12.2. Note that although Table 12.1 contains OPENGALEN ontology in the test set, the evaluation results table contains no figures about this ontology. This is due to the fact, that even in case of one source module and 20% coverage of OPENGALEN by bridge rules, the global encoding either takes unacceptably long time to load the ontology into the reasoner or the reasoner fires an exceptional situations running out of system memory.

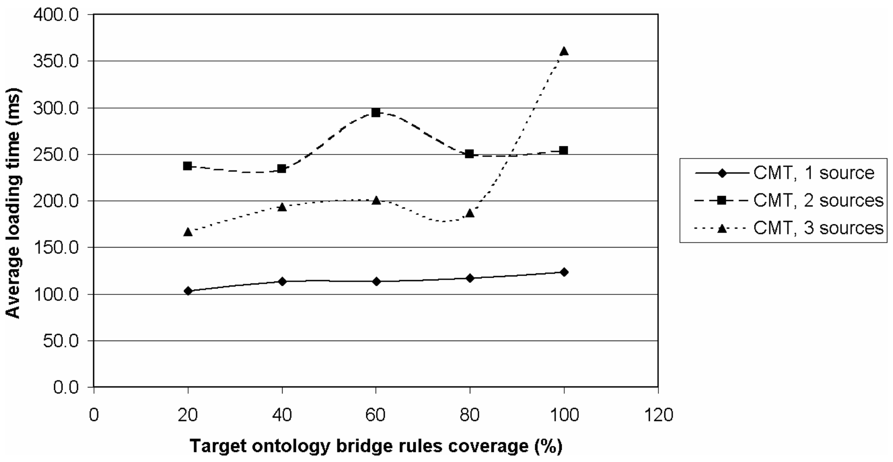
Table 12.2. Modular ontology reasoning: performance evaluation

		Ontologies															
		CMT				TRAVEL				EKAW				KA			
n sources	coverage (%)	dLoad (ms)	dSat (ms)	gLoad (ms)	gSat (ms)	dLoad (ms)	dSat (ms)	gLoad (ms)	gSat (ms)	dLoad (ms)	dSat (ms)	gLoad (ms)	gSat (ms)	dLoad (ms)	dSat (ms)	gLoad (ms)	gSat (ms)
1	20	64	14	103	8	43	17	200	8	154	20	567	6	94	18	223	7
	40	50	19	113	7	40	25	211	8	167	37	468	7	87	63	224	7
	60	50	24	113	7	50	38	217	7	167	76	440	7	73	205	341	7
	80	52	27	117	8	47	76	130	7	203	128	338	10	70	359	464	8
	100	53	37	123	7	40	125	127	9	157	167	621	9	70	736	634	8
2	20	60	17	237	7	50	16	157	6	207	20	527	8	111	22	440	8
	40	60	18	234	6	43	19	227	6	153	28	1118	7	87	28	538	6
	60	65	27	294	7	47	26	230	7	167	36	1041	7	95	44	498	7
	80	58	23	250	7	47	43	244	8	154	49	674	8	87	231	1692	7
	100	65	30	253	8	53	60	244	8	160	75	2070	8	93	182	964	9
3	20	50	20	167	7	60	18	164	6	160	20	444	7	83	19	444	7
	40	51	22	194	7	64	21	170	8	170	25	621	7	94	26	318	10
	60	53	21	201	7	57	31	294	7	161	36	724	7	85	41	354	7
	80	48	25	187	6	60	30	241	6	173	40	698	7	67	75	881	9
	100	49	31	361	7	63	39	273	7	151	56	1553	7	87	83	661	8

Let us do several observations concerning the figures in Table 12.2. Despite the change of coverage parameter or number of source modules, the *dLoad* variable slightly deviates along a certain value within each of ontologies. This value is actually equal to the loading time of a single ontology since loading of all modules



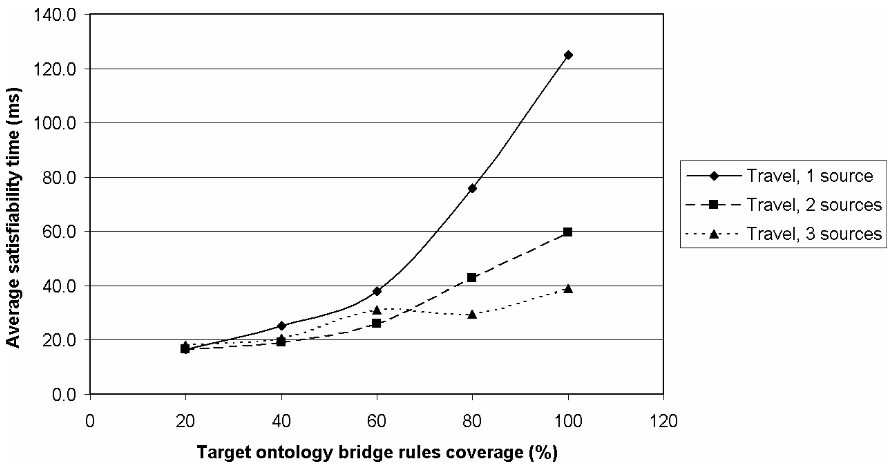
(a) Distributed reasoning



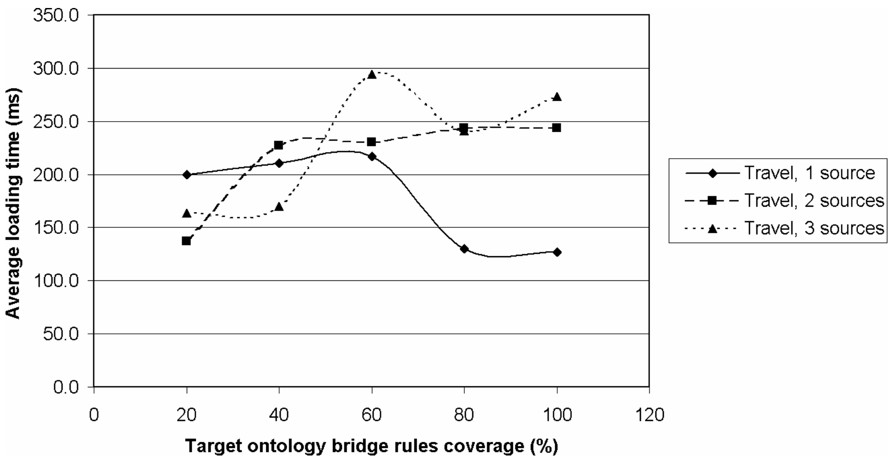
(b) Global loading

Fig. 12.10. CMT modular ontology reasoning

is done simultaneously, in parallel, by the DRAGO reasoning peers instantiated for each of the modules. Another stable variable is $gSat$. This is explained by the fact that once the ontology is preprocessed the similar reasoning requests takes a similar time to be verified. The remaining variables $dSat$ and $gLoad$ appear to be mutable in the experiments. In Figures 12.10-12.13 we graphically depict the trends of $dSat$ and $gLoad$ variables. With the increase of coverage of target ontology by bridge rules it is getting more difficult both to preprocess a global ontology, as well as it is more difficult to verify distributed satisfiability in DRAGO since more and more bridge rules actively participate in the distributed reasoning process.



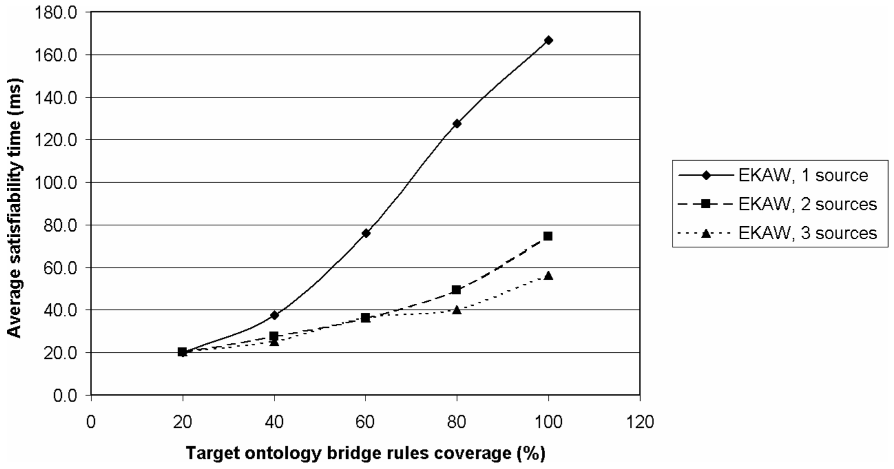
(a) Distributed reasoning



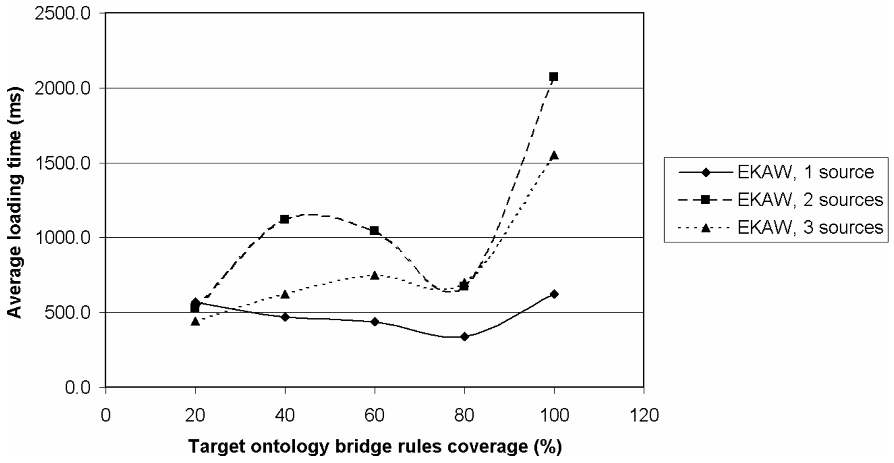
(b) Global loading

Fig. 12.11. TRAVEL modular ontology reasoning

Despite the synthetic nature of the preformed experiments and relatively small ontologies involved, the obtained figures collected in Table 12.2 still allow us to draw some trends concerning the problem of scalability of the modular ontology reasoning in the DDL framework. The application of the distributed reasoning technique allows keeping distinctly the reasoning spaces for each module in a given modular ontology. Such a distribution allows loading and preprocessing the modules by their dedicated reasoners in parallel, possibly on different computers. Such a distinction of the reasoning spaces later allows to overcome the problem of failing to load and preprocess the modular ontology as a whole by a single reasoner, as in the case of our experiments with the OPENGALEN ontology. More difficult is the configuration of a



(a) Distributed reasoning

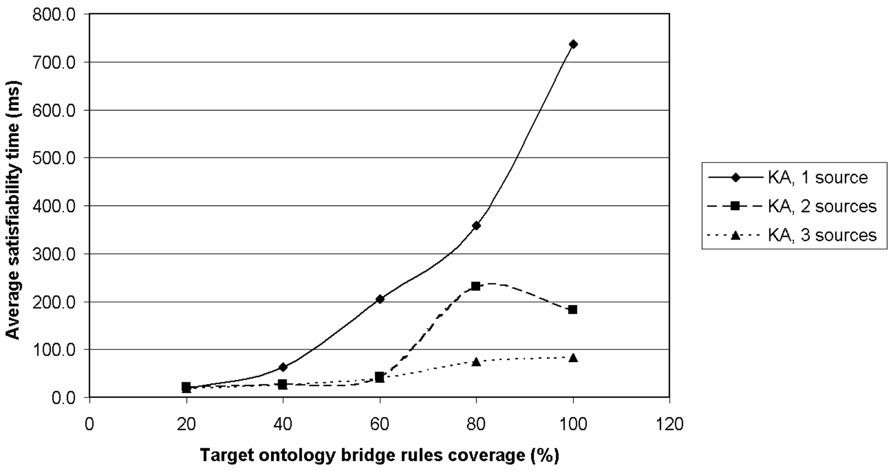


(b) Global loading

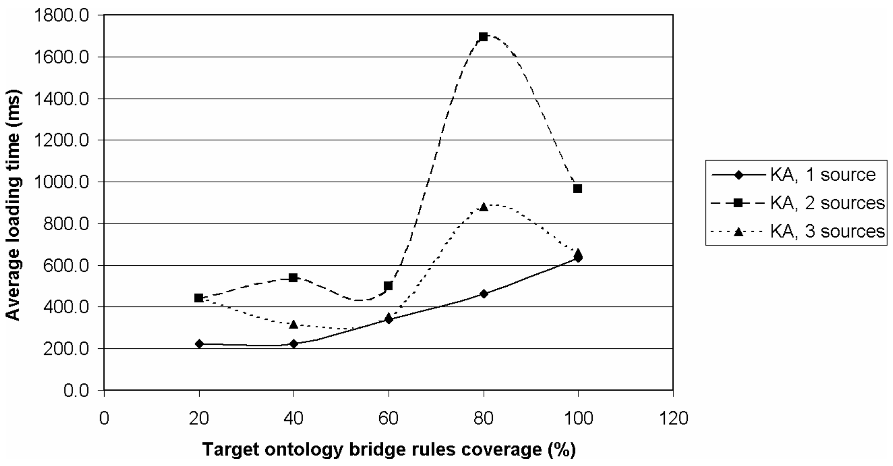
Fig. 12.12. EKAW modular ontology reasoning

modular ontology, i.e., the amount of modules as well as the amount of inter-module links — more complicated is the start of the single global reasoner compared to the simultaneous start of the several distributed local reasoners. This property of the distributed reasoner allows dealing with modular ontologies when the global reasoning approach fails to do it.

Once a modular ontology is loaded, arises the question on the responsiveness of the reasoner to the reasoning queries to be evaluated. At this point the situation with the performance of distributed and global reasoners is inverting. Indeed, the global reasoner has all the knowledge of a given modular ontology locally, while



(a) Distributed reasoning



(b) Global loading

Fig. 12.13. KA modular ontology reasoning

the distributed reasoner has to compute the knowledge possibly propagated from other modules on-the-fly, by distributed communication with corresponding module reasoners.

12.8 Conclusion

In this chapter we have presented an application of the Distributed Description Logics framework and the DRAGO reasoning system to the task of composing modular ontologies. We have shown that the bridge rules of DDL can serve as inter-module links enabling the access and import of the knowledge contained in modules. The practical feasibility of the proposed modular ontologies composition has been demonstrated by the application of the DRAGO reasoning system. Several synthetic test-cases has been generated and evaluated to see how the system treats modular ontologies of different configurations. The distributed reasoning technique of DRAGO has been confronted against the global reasoning in the standard tableau reasoner Pellet to which the whole modular ontology (containing all modules and all inter-module links) has been loaded. As we expected, the distributed reasoner allows loading such complex modular ontologies which the global reasoning technique cannot deal with. However, in the comparison with the global reasoner, the faster load of a modular ontology into the distributed reasoner sacrifices the speed of further reasoning due to the necessary distributed communications with other reasoners attached to the modules.

In this work we have considered the construction of modular ontologies from purely terminological $\mathcal{SHIQ}$ ontologies. As a future work we plan to relax this restriction and extend the results to $\mathcal{SHIQ}$ ontology modules containing individual assertions. Technically, the extension of the DDL framework concerns with admission of a new individual correspondence construct allowing to express inter-module links between individuals and to accommodate the access and import assertional knowledge contained in ontological modules.

References

1. Antoniou, G., van Harmelen, F.: Web Ontology Language: OWL. In: Handbook on Ontologies in Information Systems, pp. 67–92. Springer, Heidelberg (2004)
2. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications (2003)
3. Baader, F., Horrocks, I., Sattler, U.: Description logics as ontology languages for the semantic web. In: Hutter, D., Stephan, W. (eds.) Mechanizing Mathematical Reasoning. LNCS, vol. 2605, pp. 228–248. Springer, Heidelberg (2005)
4. Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., Andrea Stein, L.: OWL Web Ontology Language Reference. W3C Recommendation (February 2004), `http://www.w3.org/TR/owl-ref`
5. Borgida, A., Serafini, L.: Distributed Description Logics: Assimilating Information from Peer Sources. Journal of Data Semantics 1, 153–184 (2003)
6. Bouquet, P., Giunchiglia, F., van Harmelen, F., Serafini, L., Stuckenschmidt, H.: C-OWL: Contextualizing Ontologies. In: Fensel, D., Sycara, K.P., Mylopoulos, J. (eds.) ISWC 2003. LNCS, vol. 2870, pp. 164–179. Springer, Heidelberg (2003)
7. Bouquet, P., Giunchiglia, F., van Harmelen, F., Serafini, L., Stuckenschmidt, H.: Contextualizing Ontologies. Journal on Web Semantics 1(4), 325–343 (2004)
8. Ghidini, C., Giunchiglia, F.: Local Model Semantics, or Contextual Reasoning = Locality + Compatibility. Artificial Intelligence 127(2), 221–259 (2001)
9. Ghidini, C., Serafini, L.: Distributed First Order Logics. In: Proceedings of the Frontiers of Combining Systems, pp. 121–139 (2000)

10. Cuenca Grau, B., Parsia, B., Sirin, E.: Pellet: An OWL DL Reasoner. In: Proceedings of the 3rd International Semantic Web Conference (ISWC 2004) (2004), `http://www.mindswap.org/2003/pellet/`
11. Haarslev, V., Moller, R.: RACER System Description. In: Goré, R.P., Leitsch, A., Nipkow, T. (eds.) IJCAR 2001. LNCS, vol. 2083, pp. 701–706. Springer, Heidelberg (2001)
12. Horrocks, I.: Optimising Tableaux Decision Procedures for Description Logics. PhD thesis, University of Manchester (1997)
13. Horrocks, I.: Description Logics in Ontology Applications. In: Beckert, B. (ed.) TABLEAUX 2005. LNCS, vol. 3702, pp. 2–13. Springer, Heidelberg (2005)
14. Horrocks, I., Patel-Schneider, P.F.: FaCT and DLP. In: de Swart, H. (ed.) TABLEAUX 1998. LNCS, vol. 1397, pp. 27–30. Springer, Heidelberg (1998)
15. Horrocks, I., Patel-Schneider, P.F., van Harmelen, F.: From SHIQ and RDF to OWL: The Making of a Web Ontology Language. Journal of Web Semantics 1(1), 7–26 (2003)
16. Horrocks, I., Sattler, U.: A Tableaux Decision Procedure for SHOIQ. In: Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005), pp. 448–453 (2005)
17. Horrocks, I., Sattler, U., Tobies, S.: A Description Logic with Transitive and Converse Roles, Role Hierarchies and Qualifying Number Restriction. Technical Report 99-08, Technische Universität Dresden, LTCS (1999)
18. Horrocks, I., Sattler, U., Tobies, S.: Practical Reasoning for Very Expressive Description Logics. Logic Journal of IGPL 8(3), 239–263 (2000)
19. Horrocks, I., Sattler, U., Tobies, S.: Reasoning with Individuals for the Description Logic SHIQ. In: McAllester, D. (ed.) CADE 2000. LNCS, vol. 1831, pp. 482–496. Springer, Heidelberg (2000)
20. Serafini, L., Borgida, A., Tamilin, A.: Aspects of Distributed and Modular Ontology Reasoning. In: Proceedings of the 19th Joint Conference on Artificial Intelligence (IJCAI) (2005)
21. Serafini, L., Tamilin, A.: DRAGO: Distributed Reasoning Architecture for the Semantic Web. In: Gómez-Pérez, A., Euzenat, J. (eds.) ESWC 2005. LNCS, vol. 3532, pp. 361–376. Springer, Heidelberg (2005)
22. Sirin, E., Parsia, B., Cuenca Grau, B., Kalyanpur, A., Katz, Y.: Pellet: A Practical OWL-DL Reasoner. Journal of Web Semantics (2006)
23. Stuckenschmidt, H., Klein, M.: Integrity and Change in Modular Ontologies. In: Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI 2003) (2003)
24. Svab, O., Svatek, V., Berka, P., Rak, D., Tomasek, P.: OntoFarm: Towards an Experimental Collection of Parallel Ontologies. In: Poster Proceedings of the 4th International Semantic Web Conference (ISWC 2005) (2005)
25. Tessaris, S.: Questions and Answers: Reasoning and Querying in Description Logic. PhD thesis, Department of Computer Science, University of Manchester, UK (2001)
26. Tobies, S.: Complexity Results and Practical Algorithms for Logics in Knowledge Representation. PhD thesis, LuFG Theoretical Computer Science, RWTH-Aachen, Germany (2001)
27. Tsarkov, D., Horrocks, I.: FaCT++ Description Logic Reasoner: System Description. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS, vol. 4130, pp. 292–297. Springer, Heidelberg (2006)

13

Package-Based Description Logics

Jie Bao[1], George Voutsadakis[2,3], Giora Slutzki[2], and Vasant Honavar[2]

[1] Department of Computer Science, Rensselaer Polytechnic Institute, Troy, USA
baojie@cs.rpi.edu
[2] Department of Computer Science, Iowa State University, Ames, USA
{slutzki,honavar}@cs.iastate.edu
[3] Department of Computer Science, Lake Superior State University, USA
gvoutsad@lssu.edu

Summary. We present the syntax and semantics of a family of modular ontology languages, Package-based Description Logics (P-DL), to support context- specific reuse of knowledge from multiple ontology modules. In particular, we describe a P-DL $\mathcal{SHOIQP}$ that allows the *importing* of concept, role and nominal names between multiple ontology modules (each of which can be viewed as a $\mathcal{SHOIQ}$ ontology). $\mathcal{SHOIQP}$ supports contextualized interpretation, i.e., interpretation from the point of view of a specific package. We establish the necessary and sufficient conditions on domain relations (i.e., the relations between individuals in different local domains) that need to hold in order to preserve the unsatisfiability of concept formulae, monotonicity of inference, transitive reuse of knowledge across modules.

13.1 Introduction

The success of the world wide web can be partially attributed to the *network effect*: The absence of central control on the content and the organization of the web allows thousands of independent actors to contribute resources (web pages) that are interlinked to form the web. Ongoing efforts to extend the current web into a *semantic web* are aimed at enriching the web with machine interpretable content and interoperable resources and services [7]. Realizing the full potential of the semantic web requires the large-scale adoption and use of ontology-based approaches to sharing of information and resources. Constructing large ontologies typically requires collaboration among multiple individuals or groups with expertise in specific areas, with each participant contributing only a part of the ontology. Therefore, instead of a single, centralized ontology, in most application domains it is natural to have multiple distributed ontologies covering parts of the domain. Such ontologies represent the *local* knowledge of the ontology designers, i.e., knowledge that is applicable in a *context*. Because no single ontology can meet the needs of all users under every conceivable scenario, there is an urgent need for theoretically sound, yet practical, approaches that allow knowledge from multiple autonomously developed ontologies to be adapted and reused in user, context, or application-specific scenarios.

Ontologies on the semantic web need to satisfy two apparently conflicting objectives [9]:

- *Sharing* and *reuse* of knowledge across autonomously developed ontologies. An ontology may reuse another ontology by direct *importing* of selected terms in the other ontology (e.g., by referring to their URLs), or by using *mappings* between ontologies.
- The *contextuality* of knowledge or accommodation of the *local points of view*. For example, an assertion of the form "everything has the property that..." is usually made within an implicit local context which is often omitted from the statement. In fact, such a statement should be understood as "everything *in this domain* has the property that...". However, when reusing an existing ontology, the contextual nature of assertions is often neglected, leading to unintended inferences.

OWL adopts an importing mechanism to support integration of ontology modules. However, the importing mechanism in OWL, implemented by the `owl:imports` construct, in its current form, suffers from several serious drawbacks: (a) It directly introduces both terms and axioms of the imported ontologies into the importing ontology, and thus fails to support contextual reuse; (b) It provides no support for partial reuse of an ontology module.

Consequently, there have been several efforts aimed at developing formalisms that allow *reuse* of knowledge from multiple ontologies via *contextualized interpretations* in multiple local domains instead of a single shared global interpretation domain. Contextualized reuse of knowledge requires the interactions between local interpretations to be controlled. Examples of such modular ontology languages include: Distributed Description Logics (DDL) [8], $\mathcal{E}$-Connections [16] and Semantic Importing [20].

An alternative approach to knowledge reuse is based on the notion of *conservative extension* [12, 13, 14, 15], which allows ontology modules to be interpreted using standard semantics by requiring that they share the same global interpretation domain. To avoid undesired effects from combining ontology modules, this approach requires that such a combination be a conservative extension of component modules. More precisely, if O is the union of a set of ontology modules $\{O_1, ..., O_n\}$, then we say O is a conservative extension of O_i if $O \models \alpha \Leftrightarrow O_i \models \alpha$, for any α in the language of O_i. This guarantees that combining knowledge from several ontology modules does not alter the consequences of knowledge contained in any component module. Thus, a combination of ontology modules cannot induce a new concept inclusion relation between concepts expressible in any of the component modules.

Current approaches to knowledge reuse have several limitations. To preserve contextuality, existing modular ontology languages offer only limited ways to connect ontology modules and, hence, limited ability to reuse knowledge across modules. For instance, DDL does not allow concept construction using foreign roles or concepts. $\mathcal{E}$-Connections, on the other hand, does not allow concept subsumptions across ontology modules or the use of foreign roles. Finally, Semantic Importing, in its current form, only allows each component module to be in $\mathcal{ALC}$. None of the existing approaches supports knowledge reuse in a setting where each ontology module uses a representation language that is as expressive as OWL-DL, i.e., $\mathcal{SHOIN}(D)$.

Furthermore, some of the existing modular ontology languages suffer from reasoning difficulties that can be traced back to the absence of natural ways to restrict the relations between individuals in different local domains. For example, DDL does not support the transitivity of inter-module concept subsumptions (known as *bridge rules*) in general. Moreover, in DDL a concept that is declared as being more specific than two disjoint concepts in another module may still be satisfiable (the inter-module satisfiability problem) [3, 16]. Undisciplined use of generalized links in $\mathcal{E}$-Connections has also been shown to lead to reasoning difficulties [2].

Conservative extensions [13, 14, 15], in their current form, require a single global interpretation domain and, consequently, prevent different modules from interpreting axioms within their own local contexts. Hence, the designers of different ontology modules have to anticipate all possible contexts in which knowledge from a specific module might be reused. As a result, several modeling scenarios that would, otherwise, be quite useful in practice, such as the refinement of relations between existing concepts in an ontology module and the general reuse of nominals [19], are precluded.

Against this background, this chapter, building on previous work of a majority of the authors [3], develops a formalism that can support *contextual* reuse of knowledge from multiple ontology modules. The resulting modular ontology language, Package-based Description Logic (P-DL) $\mathcal{SHOIQP}$:

- Allows each ontology module to use a subset of $\mathcal{SHOIQ}$ [17], i.e., $\mathcal{ALC}$ augmented with transitive roles, role inclusion, role inversion, qualified number restriction and nominal concepts and, hence, covers a significant fragment of OWL-DL.
- Supports more flexible modeling scenarios than those supported by existing approaches through a mechanism of *semantic importing* of names (including concept, role and nominal names) across ontology modules[1].
- Contextualizes the interpretation of reused knowledge. Locality of axioms in ontology modules is obtained "for free" by its *contextualized semantics*, thereby freeing ontology engineers from the burden of ensuring the reusability of an ontology module in contexts that are hard to foresee when constructing the module. A natural consequence of contextualized interpretation is that inferences are always drawn *from the point of view* of a *witness* module. Thus, different modules might infer different consequences, based on the knowledge that they import from other modules.

[1] Note that importing in OWL, implemented by the `owl:imports` is essentially syntactic in nature. The difference between syntactic importing and semantic importing is best illustrated by an analogy with the writing of scientific articles: Knowledge reuse via `owl:imports` analogous to *cut and paste* from a source article; In contrast, semantic importing is akin to knowledge reuse by means of *citation* of source article.

- Ensures that the results of reasoning are always the same as those obtained by a standard reasoner over an integrated ontology resulting from combining the relevant knowledge in a context-specific manner. Thus, unlike in the case of DDL and Semantic Importing of Pan et al., P-DL ensures the *monotonicity* of inference in the distributed setting.
- Avoids several of the known reasoning difficulties of the existing approaches, e.g., lack of support for transitive reusability and nonpreservation of concept unsatisfiability.

13.2 Semantic Importing

This section introduces the syntax and semantics of the proposed language $\mathcal{SHOIQP}$. We will use a simple example shown in Figure 13.1 to illustrate some of the basic features of the P-DL syntax.

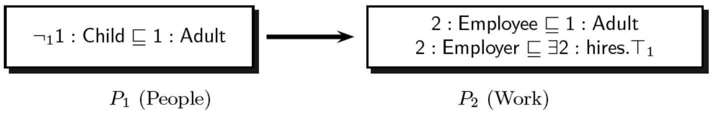

$\neg_1 1:\mathsf{Child} \sqsubseteq 1:\mathsf{Adult}$
$2:\mathsf{Employee} \sqsubseteq 1:\mathsf{Adult}$
$2:\mathsf{Employer} \sqsubseteq \exists 2:\mathsf{hires}.\top_1$
P_1 (People)
P_2 (Work)

Fig. 13.1. Semantic Importing

13.2.1 Syntax

Packages

Informally, a package in $\mathcal{SHOIQP}$ can be viewed as a $\mathcal{SHOIQ}$ TBox and RBox. For example, in Figure 13.1 there are two packages, package P_1 describes the domain of People and P_2 describes the domain of Work.

We define the *signature* $\mathsf{Sig}(P_i)$ of a package P_i as the set of names used in P_i. $\mathsf{Sig}(P_i)$ is the disjoint union of the set of concept names NC_i, the set of role names NR_i and the set of nominal names NI_i used in package P_i. The set of roles in P_i is defined as $\overline{\mathsf{NR}}_i = \mathsf{NR}_i \cup \{R^-|R \in \mathsf{NR}_i\}$ where R^- is the *inverse* of the role name R.

The signature $\mathsf{Sig}(P_i)$ of package P_i is divided into two disjoint parts: its local signature $\mathsf{Loc}(P_i)$ and its external signature $\mathsf{Ext}(P_i)$. Thus, in the example shown in Figure 13.1, $\mathsf{Sig}(P_2) = \{\mathsf{Employee}, \mathsf{Adult}, \mathsf{Employer}, \mathsf{hires}\}$; $\mathsf{Loc}(P_2) = \{\mathsf{Employee}, \mathsf{Employer}, \mathsf{hires}\}$; and $\mathsf{Ext}(P_2) = \{\mathsf{Adult}\}$.

For all $t \in \mathsf{Loc}(P_i)$, P_i (and only P_i) is the *home package* of t, denoted by $P_i = \mathsf{Home}(t)$, and t is called an *i-name* (more specifically, an *i-concept name*, an *i-role name*, or an *i-nominal name*). We will use "$i : X$" to denote an i-name X and may drop the prefix when it is clear from the context. We use *i-role* to refer to an i-role name or its *inverse*. In the example shown in Figure 13.1, the home package of the terms Child and Adult is P_1 (People); and that of Employee, Employer and hires is P_2 (Work).

A role name $R \in \mathsf{NR}_i$ may be declared to be *transitive* in P_i using an axiom $\mathsf{Trans}_i(R)$. If R is declared transitive, R^- is also said to be *transitive*. We use $\mathsf{Tr}_i(R)$ to denote a role R being transitive in P_i.

A *role inclusion* axiom in P_i is an expression of the form $R \sqsubseteq S$, where R and S are i-roles. The *role hierarchy* for P_i is the set of all role inclusion axioms in P_i. The RBox $\mathcal{R}_i$ consists of the role hierarchy $\mathbf{R}_i$ for P_i and the set of role transitivity declarations $\mathsf{Trans}_i(R)$. For a role hierarchy $\mathbf{R}_i$, if $R \sqsubseteq S \in \mathbf{R}_i$, then R is called a *sub-role* of S and S is called a *super-role* of R *w.r.t.* $\mathbf{R}_i$. An i-role is called *locally simple* if it neither transitive nor has any transitive sub-role in P_i.

The set of $\mathcal{SHOIQP}$ concepts in P_i is defined inductively by the following grammar:

$$C := A|o|\neg_k C|C \sqcap C|C \sqcup C|\forall R.C|\exists R.C|(\leq nS.C)|(\geq nS.C)$$

where $A \in \mathsf{NC}_i$, $o \in \mathsf{NI}_i$, n is a non-negative integer, $R \in \overline{\mathsf{NR}}_i$, and $S \in \overline{\mathsf{NR}}_i$ is a locally simple role; $\neg_k C$ denotes the *contextualized negation* of concept *C w.r.t.* P_k. For any *k* and k-concept name C, $\top_k = \neg_k C \sqcup C$, and $\bot = \neg_k C \sqcap C$. Thus, there is no universal top ($\top$) concept or global negation ($\neg$). Instead, we have for each package P_k, a contextualized top $\top_k$ and a contextualized negation $\neg_k$. This allows a logical formula in P-DL (including $\mathcal{SHOIQP}$) to be interpreted within the context of a specific package. Thus, in the example shown in Figure 13.1, $\neg_1 1:\mathsf{Child}$ in P_1 describes only the individuals *in the domain of People* that are not not children (that is, *not* $1:\mathsf{Child}$).

A *general concept inclusion* (GCI) *axiom* in P_i is an expression of the form $C \sqsubseteq D$, where C, D are concepts in P_i. The TBox $\mathcal{T}_i$ of P_i is the set of GCIs in P_i. Thus, formally, a *package* P_i is a pair $P_i := \langle \mathcal{T}_i, \mathcal{R}_i \rangle$. A $\mathcal{SHOIQP}$ ontology Σ is a set of packages $\{P_i\}$. We assume that every name used in a $\mathcal{SHOIQP}$ ontology Σ has a home package in Σ.

Semantic Importing between Packages

If a concept, role or nominal name $t \in \mathsf{Loc}(P_j) \cap \mathsf{Ext}(P_i)$, $i \neq j$, we say that P_i *imports* t and denote it as $P_j \xrightarrow{t} P_i$. We require that transitivity of roles be preserved under importing. Thus, if $P_j \xrightarrow{R} P_i$ where R is a j-role name, then $\mathsf{Trans}_i(R)$ iff $\mathsf{Trans}_j(R)$. If any local name of P_j is imported into P_i, we say that P_i imports P_j and denote it by $P_j \mapsto P_i$. In the example shown in Figure 13.1, P_2 imports P_1.

The *importing transitive closure* of a package P_i, denoted by P_i^+, is the set of all packages that are directly or indirectly imported by P_i. That is, P_i^+ is the smallest subset of $\{P_i\}$, such that

- $\forall j \neq i, P_j \mapsto P_i \Rightarrow P_j \in P_i^+$
- $\forall k \neq j \neq i, (P_k \mapsto P_j) \wedge (P_j \in P_i^+) \Rightarrow P_k \in P_i^+$

Let $P_i^* = \{P_i\} \cup P_i^+$. A $\mathcal{SHOIQP}$ ontology $\Sigma = \{P_i\}$ has an *acyclic importing relation* if, for all i, $P_i \notin P_i^+$; otherwise, it has a *cyclic importing relation*. The importing relation in the example in Figure 13.1 is acyclic.

We denote a Package-based Description Logic (P-DL) by adding the letter $\mathcal{P}$ to the notation for the corresponding DL. For example, $\mathcal{ALCP}$ is the package extension of the DL $\mathcal{ALC}$. We denote by $\mathcal{P}_\mathcal{C}$ a restricted type of P-DL that only allows importing of concept names. $\mathcal{P}^-$ denotes a P-DL with acyclic importing. In particular, $\mathcal{ALCP}_\mathcal{C}^-$ was studied in [1], $\mathcal{ALCP}_\mathcal{C}$ was studied in [4] and $\mathcal{SHOIQP}$ was studied in [5]. The example in Figure 13.1 is in $\mathcal{ALCP}_\mathcal{C}^-$.

Syntax Restrictions on Semantic Importing

Restrictions on Negations. We require that $\neg_k C$ (hence also $\top_k$) can appear in P_i, $i \neq k$, only if $P_k \mapsto P_i$. Intuitively, this means that k-negation can appear only in P_k or any package that directly imports P_k.

Restrictions on Imported Role Names. We require that *an imported role should not be used in role inclusion axioms*. This restriction is imposed because of two reasons. First, decidability requires that a role that is used in number restrictions be "globally" simple, i.e., that it has no transitive sub-role across any importing chain[2] [18]. In practice, it is useful to restrict the use of imported roles in such a way that a role is globally simple iff it is locally simple. Second, a reduction of $\mathcal{SHOIQP}$ without such a restriction to an integrated ontology may require some features that are beyond the expressivity of $\mathcal{SHOIQ}$, such as role intersection.

$\mathcal{SHOIQP}$ Examples

The semantic importing approach described here can model a broad range of scenarios that can also be modeled using existing approaches.

Example 1. Inter-module concept and role inclusions. *Suppose we have a* people *ontology P_1:*

$$\begin{aligned}
\neg_1 1:\mathsf{Man} &\sqsubseteq 1:\mathsf{Woman}\\
1:\mathsf{Man} &\sqsubseteq 1:\mathsf{People}\\
1:\mathsf{Woman} &\sqsubseteq 1:\mathsf{People}\\
1:\mathsf{Boy} \sqcup 1:\mathsf{Girl} &\sqsubseteq 1:\mathsf{Child}\\
1:\mathsf{Husband} &\sqsubseteq 1:\mathsf{Man} \sqcap \exists 1:\mathsf{marriedTo}.1:\mathsf{Woman}
\end{aligned}$$

Suppose the Work *ontology P_2 imports some of the knowledge from the people ontology:*

$$2:\mathsf{Employee} \sqsubseteq 1:\mathsf{People} \tag{13.1}$$

$$2:\mathsf{Employer} \equiv \exists 2:\mathsf{hires}.1:\mathsf{People} \tag{13.2}$$

$$1:\mathsf{Child} \sqsubseteq \neg_2 2:\mathsf{Employee} \tag{13.3}$$

$$2:\mathsf{EqualOpportunityEmployer} \sqsubseteq \exists 2:\mathsf{hires}.1:\mathsf{Man} \sqcap \exists 2:\mathsf{hires}.1:\mathsf{Woman} \tag{13.4}$$

Axioms (13.1) *models inter-module concept inclusion. This example also illustrates that the semantic importing approach can realize concept specialization (Axiom* (13.2)*) and generalization (Axiom* (13.3)*).*

[2] This follows from the reduction from $\mathcal{SHOIQP}$ to $\mathcal{SHOIQ}$ given in Section 13.3.

Example 2. Use of foreign roles or foreign concepts to construct local concepts. *Suppose a* marriage *ontology P_3 reuses the people ontology:*

$$(= 1\ (1 : \mathsf{marriedTo}).(1 : \mathsf{Woman})) \sqsubseteq 3 : \mathsf{Monogamist} \tag{13.5}$$

$$3 : \mathsf{MarriedPerson} \sqsubseteq \forall(1 : \mathsf{marriedTo}).(3 : \mathsf{MarriedPerson}) \tag{13.6}$$

$$3 : \mathsf{NuclearFamily} \sqsubseteq \exists(3 : \mathsf{hasMember}).(1 : \mathsf{Child}) \tag{13.7}$$

A complex concept in P_3 may be constructed using an imported role (13.6)*, an imported concept* (13.7)*, or both an imported role and an imported concept* (13.5).

Example 3. The use of nominals. *Suppose the work ontology P_2, defined above, is augmented with additional knowledge from a* calendar *ontology P_4, to obtain an augmented work ontology. Suppose P_4 contains the following axiom:*

$$4{:}\mathsf{WeekDay} = \{4{:}Mon, 4{:}Tue, 4{:}Wed, 4{:}Thu, 4{:}Fri\},$$

where the nominals are shown in italic font. Suppose the new version of P_2 contains the following additional axioms:

$$4 : Fri \sqsubseteq \exists(2 : \mathsf{hasDressingCode}).(2 : \mathsf{CasualDress})$$
$$\top_2 \sqsubseteq \exists(2 : \mathsf{hasDressingCode}^-).(4 : \mathsf{WeekDay})$$

13.2.2 Semantics

A $\mathcal{SHOIQP}$ ontology has *localized semantics* in the sense that each package has its own local interpretation domain. Formally, for a $\mathcal{SHOIQP}$ ontology $\Sigma = \{P_i\}$, a *distributed interpretation* is a tuple $\mathcal{I} = \langle\{\mathcal{I}_i\}, \{r_{ij}\}_{P_i \in P_j^+}\rangle$, where $\mathcal{I}_i$ is a *local interpretation* of package P_i, with (a not necessarily non-empty) domain $\Delta^{\mathcal{I}_i}$, $r_{ij} \subseteq \Delta^{\mathcal{I}_i} \times \Delta^{\mathcal{I}_j}$ is the *(image) domain relation* for the interpretation of the direct or indirect importing relation from P_i to P_j. For convenience, we use $r_{ii} = \mathrm{id}_{\Delta^{\mathcal{I}_i}} := \{(x,x)|x \in \Delta^{\mathcal{I}_i}\}$ to denote the identity mapping in the local domain $\Delta^{\mathcal{I}_i}$. Taking this convention into account, the distributed interpretation $\mathcal{I} = \langle\{\mathcal{I}_i\}, \{r_{ij}\}_{P_i \in P_j^+}\rangle$ may also be denoted by $\mathcal{I} = \langle\{\mathcal{I}_i\}, \{r_{ij}\}_{P_i \in P_j^*}\rangle$.

To facilitate our further discussion of interpretations, the following notational conventions will be used throughout. Given i, j, such that $P_i \in P_j^*$, for every $x \in \Delta^{\mathcal{I}_i}$, $A \subseteq \Delta^{\mathcal{I}_i}$ and $S \subseteq \Delta^{\mathcal{I}_i} \times \Delta^{\mathcal{I}_i}$, define[3] (please see Figure 13.2 and 13.3 for illustration):

$$\begin{aligned}
r_{ij}(A) &= \{y \in \Delta^{\mathcal{I}_j} | \exists x \in A, (x,y) \in r_{ij}\}, && \text{(concept image)}\\
r_{ij}(S) &= r_{ij} \circ S \circ r_{ij}^- && \text{(role image)}\\
&= \{(z,w) \in \Delta^{\mathcal{I}_j} \times \Delta^{\mathcal{I}_j} | \exists (x,y) \in S, (x,z) \in r_{ij} \wedge (y,w) \in r_{ij}\}, &&\\
S(x) &= \{y \in \Delta^{\mathcal{I}_i} | (x,y) \in S\} && \text{(successor set)}
\end{aligned}$$

[3] In this chapter, $f_1 \circ ... \circ f_n$ denotes the composition of n relations $f_1, ..., f_n$, i.e., $(f_1 \circ ... \circ f_n)(x) = f_1(...f_n(x))$.

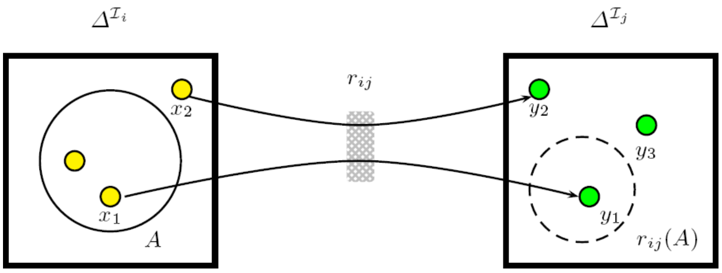


Fig. 13.2. Concept Image

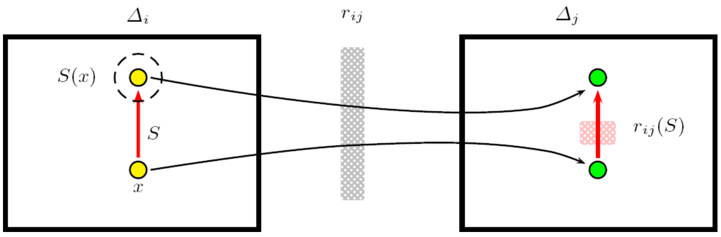


Fig. 13.3. Successor Set and Role Image

Moreover, let ρ be the equivalence relation on $\bigcup_i \Delta^{\mathcal{I}_i}$ generated by the collection of ordered pairs $\bigcup_{P_i \in P_j^*} r_{ij}$. This is the symmetric and transitive closure of the set $\bigcup_{P_i \in P_j^*} r_{ij}$. Define, for every i, j, $\rho_{ij} = \rho \cap (\Delta^{\mathcal{I}_i} \times \Delta^{\mathcal{I}_j})$.

Each of the local interpretations $\mathcal{I}_i = \langle \Delta^{\mathcal{I}_i}, \cdot^{\mathcal{I}_i} \rangle$ consists of a domain $\Delta^{\mathcal{I}_i}$ and an interpretation function $\cdot^{\mathcal{I}_i}$, which maps every concept name to a subset of $\Delta^{\mathcal{I}_i}$, every role name to a subset of $\Delta^{\mathcal{I}_i} \times \Delta^{\mathcal{I}_i}$ and every nominal name to an element in $\Delta^{\mathcal{I}_i}$. We require that the interpretation function $\cdot^{\mathcal{I}}$ satisfies the following equations, where R is a j-role, S is a locally simple j-role, C, D are concepts:

$$\begin{aligned}
R^{\mathcal{I}_i} &= (R^{\mathcal{I}_i})^+, \text{ if } \mathsf{Trans}_i(R) \in \mathcal{R}_i \\
(R^-)^{\mathcal{I}_i} &= \{(x,y) | (y,x) \in R^{\mathcal{I}_i}\} \\
(C \sqcap D)^{\mathcal{I}_i} &= C^{\mathcal{I}_i} \cap D^{\mathcal{I}_i} \\
(C \sqcup D)^{\mathcal{I}_i} &= C^{\mathcal{I}_i} \cup D^{\mathcal{I}_i} \\
(\neg_j C)^{\mathcal{I}_i} &= r_{ji}(\Delta^{\mathcal{I}_j}) \backslash C^{\mathcal{I}_i} \\
(\exists R.C)^{\mathcal{I}_i} &= \{x \in r_{ji}(\Delta^{\mathcal{I}_j}) | \exists y \in \Delta^{\mathcal{I}_i}, (x,y) \in R^{\mathcal{I}_i} \wedge y \in C^{\mathcal{I}_i}\} \\
(\forall R.C)^{\mathcal{I}_i} &= \{x \in r_{ji}(\Delta^{\mathcal{I}_j}) | \forall y \in \Delta^{\mathcal{I}_i}, (x,y) \in R^{\mathcal{I}_i} \to y \in C^{\mathcal{I}_i}\} \\
(\geqslant nS.C)^{\mathcal{I}_i} &= \{x \in r_{ji}(\Delta^{\mathcal{I}_j}) | \, |\{y \in \Delta^{\mathcal{I}_i} | (x,y) \in S^{\mathcal{I}_i} \wedge y \in C^{\mathcal{I}_i}\}| \geqslant n\} \\
(\leqslant nS.C)^{\mathcal{I}_i} &= \{x \in r_{ji}(\Delta^{\mathcal{I}_j}) | \, |\{y \in \Delta^{\mathcal{I}_i} | (x,y) \in S^{\mathcal{I}_i} \wedge y \in C^{\mathcal{I}_i}\}| \leqslant n\}
\end{aligned}$$

Note that, when $i = j$, since $r_{ii} = \mathrm{id}_{\Delta^{\mathcal{I}_i}}$, $(\neg_j C)^{\mathcal{I}_i}$ reduces to the usual negation $(\neg_i C)^{\mathcal{I}_i} = \Delta^{\mathcal{I}_i} \backslash C^{\mathcal{I}_i}$. Similarly, the other semantic definitions also reduce to the usual DL semantic definitions.

For an example of contextualized negation, suppose $A = C^{\mathcal{I}_i}$ in the Figure 13.2, then $(\neg_i C)^{\mathcal{I}_j}$ will only contain y_2 but not y_3. On the other hand, $(\neg_j C)^{\mathcal{I}_j}$ is will contain both y_2 and y_3.

A local interpretation $\mathcal{I}_i$ *satisfies* a role inclusion axiom $R_1 \sqsubseteq R_2$ iff $R_1^{\mathcal{I}_i} \subseteq R_2^{\mathcal{I}_i}$ and a GCI $C \sqsubseteq D$ iff $C^{\mathcal{I}_i} \subseteq D^{\mathcal{I}_i}$. $\mathcal{I}_i$ is a *model* of P_i, denoted by $\mathcal{I}_i \vDash P_i$, if it satisfies all axioms in P_i.

The proposed semantics of $\mathcal{SHOIQP}$ is motivated by the need to overcome some of the limitations of existing approaches that can be traced back to the arbitrary construction of domain relations and the lack of support for contextualized interpretation. Specifically, we seek a semantics that satisfies the following desiderata:

- **Preservation of concept unsatisfiability.** The intuition is that an unsatisfiable concept expression should never be reused so as to be interpreted as a satisfiable concept. Formally, we say that a domain relation r_{ij} *preserves the unsatisfiability* of a concept C, that appears in both P_i and P_j, if whenever $C^{\mathcal{I}_i} = \emptyset$, it is necessarily the case that $C^{\mathcal{I}_j} = \emptyset$.
- **Transitive reusability of knowledge.** The intention is that the consequences of some of the axioms in one module can be propagated in a transitive fashion to other ontology modules. For example, if a package P_i asserts that $C \sqsubseteq D$, and P_j directly or indirectly imports that axiom from P_i, then it should be the case that $C \sqsubseteq D$ is also valid from the *point of view* of P_j.
- **Contextualized interpretation of knowledge.** The idea is that the interpretation of assertions in each ontology module is constrained by their context. When knowledge, e.g., axioms, in that module is reused by other modules, the interpretation of the reused knowledge should be constrained by the context in which the knowledge is being reused.
- **Improved expressivity.** Ideally, the language should support
  1. both inter-module concept inclusion and concept construction using foreign concepts, roles and nominals;
  2. more general reuse of roles and of nominals than allowed by existing approaches.

A major goal of this chapter is to explore the constraints that need to be imposed on local interpretations so that the resulting semantics for $\mathcal{SHOIQP}$ satisfies the desiderata enumerated above. These constraints are presented in the following:

Definition 1. *An interpretation* $\mathcal{I} = \langle \{\mathcal{I}_i\}, \{r_{ij}\}_{P_i \in P_j^*} \rangle$ *is a* model *of a* $\mathcal{SHOIQP}$ *KB* $\Sigma = \{P_i\}$*, denoted as* $\mathcal{I} \vDash \Sigma$*, if* $\bigcup_i \Delta^{\mathcal{I}_i} \neq \emptyset$*, i.e., at least one of the local interpretation domains is non-empty*[4]*, and the following conditions are satisfied:*

[4] This agrees with conventional model-theoretic semantics, where an ordinary model (of a single package) is assumed to have a non-empty domain.

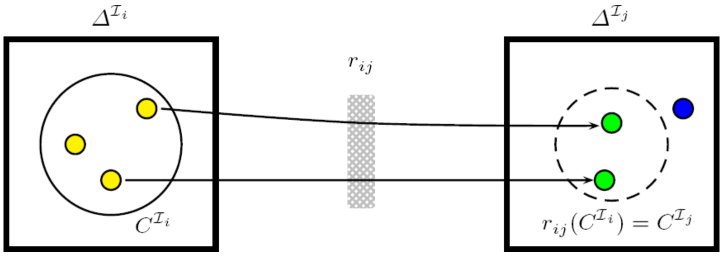
1. *For all i, j, r_{ij} is one-to-one, i.e., it is an injective partial function.*
2. Compositional Consistency*: For all i, j, k s.t. $P_i \in P_k^*$ and $P_k \in P_j^*$, we have $\rho_{ij} = r_{ij} = r_{kj} \circ r_{ik}$.*
3. *For every i-concept name C that appears in P_j, we have $r_{ij}(C^{\mathcal{I}_i}) = C^{\mathcal{I}_j}$.*
4. *For every i-role R that appears in P_j, we have $R^{\mathcal{I}_j} = r_{ij}(R^{\mathcal{I}_i})$.*
5. Cardinality Preservation for Roles*: For every i-role R that appears in P_j and every $(x, x') \in r_{ij}$, $y \in R^{\mathcal{I}_i}(x)$ iff $r_{ij}(y) \in R^{\mathcal{I}_j}(x')$.*
6. *For every i-nominal o that appears in P_j, $(o^{\mathcal{I}_i}, o^{\mathcal{I}_j}) \in r_{ij}$.*
7. *$\mathcal{I}_i \vDash P_i$, for every i.*

The proposed semantics for $\mathcal{SHOIQP}$ is an extension of the semantics for $\mathcal{ALCP_C}$ [4], which uses Conditions 1,2,3 and 7 above, and borrows Condition 5 from the semantics of Semantic Importing [20].

Intuitively, one-to-oneness (Condition 1, see Figure 13.4) and compositional consistency (Condition 2, Figure 13.5) ensure that the parts of local domains connected by domain relations match perfectly. Conditions 3 and 4 ensure consistency between the interpretations of concepts and of roles in their home package and the interpretations in the packages that import them. Condition 5 (Figure 13.6) ensures that r_{ij} is a total bijection from $R^{\mathcal{I}_i}(x)$ to $R^{\mathcal{I}_j}(r_{ij}(x))$. In particular, the sizes $|R^{\mathcal{I}_i}(x)|$ and $|R^{\mathcal{I}_j}(r_{ij}(x))|$ are always equal in different local domains. Condition 6 ensures the uniqueness of nominals. In Section 4, we will show that Conditions 1-7 are minimally sufficient to guarantee that the desiderata for the semantics of $\mathcal{SHOIQP}$ as outlined above are indeed satisfied.

Note that Condition 2 implies that if P_i and P_j mutually (possibly indirectly) import one another, then $r_{ij} = \rho_{ij} = \rho_{ji}^{-} = r_{ji}^{-}$ and r_{ij} is a total function from $\Delta^{\mathcal{I}_i}$ to $\Delta^{\mathcal{I}_j}$. However, if $P_j \notin P_i^*$, r_{ji} does not necessarily exist even if r_{ij} exists. In that case, r_{ij} is not necessarily a total function.

Definition 2. *An ontology Σ is* consistent as witnessed by a package P_w *of Σ if P_w^* has a model $\mathcal{I} = \langle \{\mathcal{I}_i\}, \{r_{ij}\}_{P_i \in P_j^+} \rangle$, such that $\Delta^{\mathcal{I}_w} \neq \emptyset$. A concept C is* satisfiable as witnessed by P_w *if there is a model $\mathcal{I}$ of P_w^*, such that $C^{\mathcal{I}_w} \neq \emptyset$. A concept*



An image domain relation in P-DL is one-to-one, i.e., it is a partial injective function. It is not necessarily total, i.e., some individuals of $C^{\mathcal{I}_i}$ may not be mapped to $\Delta^{\mathcal{I}_j}$.

Fig. 13.4. One-to-One Domain Relation

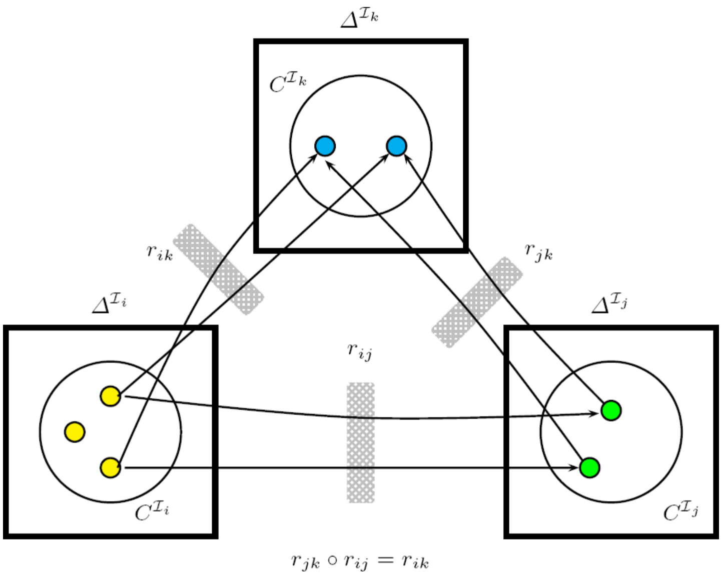
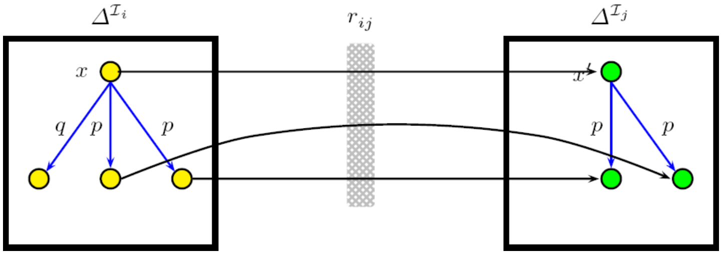


Fig. 13.5. Compositionally Consistent Domain Relation



If an i-role p is imported by P_j, then every pair of p instances must have a "preimage" pair in Δ_i. The cardinality preservation condition for roles, illustrated in this figure, requires that, if an individual x in $\Delta^{\mathcal{I}_i}$ has an image individual x' in $\Delta^{\mathcal{I}_j}$, then each of its p-neighbors must have an image in $\Delta^{\mathcal{I}_j}$ which is a p-neighbor of x'.

Fig. 13.6. Cardinality Preservation for Roles

subsumption $C \sqsubseteq D$ is valid as witnessed by P_w*, denoted by* $C \sqsubseteq_w D$*, if, for every model* $\mathcal{I}$ *of* P_w^**,* $C^{\mathcal{I}_w} \subseteq D^{\mathcal{I}_w}$.

Hence, in $\mathcal{SHOIQP}$, the questions of consistency, satisfiability and subsumption are always answered from the local point of view of a *witness package* and it is possible that different packages draw different conclusions from their own points of view.

The following examples show some inference problems that a P-DL ontology can tackle. Precise proofs for general cases will be given in Section 13.4.

Example 4. *Transitive subsumption propagation. Given three packages:* $P_1 : \{1 : A \sqsubseteq 1 : B\}$*,* $P_2 : \{1 : B \sqsubseteq 2 : C\}$*,* $P_3 : \{2 : C \sqsubseteq 3 : D\}$*, the subsumption query* $1 : A \sqsubseteq 3 : D$ *is answered in the affirmative as witnessed by* P_3*.*

Example 5. *Detection of inter-module unsatisfiability. Given two packages* $P_1 : \{1 : B \sqsubseteq 1 : F\}$*,* $P_2 : \{1 : P \sqsubseteq 1 : B, 2 : P \sqsubseteq \neg 1 : F\}$*,* $2 : P$ *is unsatisfiable as witnessed by* P_2*.*

Example 6. *Reasoning from a local point of view. Given two packages* $P_1 : \{1 : A \sqsubseteq 1 : C\}$*,* $P_2 : \{1 : A \sqsubseteq \exists 2 : R.(2 : B), 2 : B \sqsubseteq 1 : A \sqcap (\neg 1 : C)\}$*, consider the satisfiability of* $1 : A$ *as witnessed by* P_1 *and* P_2*, respectively. It is easy to see* A *is satisfiable when witnessed by* P_1*, but unsatisfiable when witnessed by* P_2*. Thus, inferences in P-DL are always drawn from the point of view of a witness package. Different witnesses, because they operate on different domains, and have access to different pieces of knowledge, can draw dramatically different conclusions.*

Discussion: Relation between the Semantics of P-DL and Partially-Overlapping Local Domain Semantics

In [10] a semantics based on partially overlapping domains was proposed for terminology mappings between ontology modules. In that framework, a global interpretation $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ is given together with local domains $\Delta^{\mathcal{I}_i}$, that are subsets of $\Delta^{\mathcal{I}}$. Any two local domains may be partially overlapping. Moreover, inclusions between concepts are of the following two forms:

- $i : C \sqsubseteq_{\text{ext}} j : D$ (extensional inclusion), with semantics $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, and
- $i : C \sqsubseteq_{\text{int}} j : D$ (intentional inclusion), with semantics $C^{\mathcal{I}} \cap \Delta^{\mathcal{I}_i} \cap \Delta^{\mathcal{I}_j} \subseteq D^{\mathcal{I}} \cap \Delta^{\mathcal{I}_i} \cap \Delta^{\mathcal{I}_j}$.

Since P-DL semantics does not envision a global point of view, extensional inclusion has no corresponding notion in P-DL semantics. In addition, P-DL semantics differs significantly from this approach in that, while both intentional and extensional inclusions are not directional, the semantic importing in P-DL is. To make this distinction clearer, consider two packages P_i and P_j, such that $P_i \mapsto P_j$. Let C, D be two i-concept names that are imported by P_j and consider the interpretation where $\Delta^{\mathcal{I}_i} = \{x, y, z\}, \Delta^{\mathcal{I}_j} = \{y, z\}$, $C^{\mathcal{I}_i} = \{x, y\}, D^{\mathcal{I}_i} = \{y, z\}$ and $r_{ij} = \{\langle y, y\rangle, \langle z, z\rangle\}$. Then, in P-DL, from the point of view of package P_i, we have $C^{\mathcal{I}_i} = \{x, y\} \not\subseteq \{y, z\} = D^{\mathcal{I}_i}$. Therefore, $\mathcal{I} \not\models_i C \sqsubseteq D$. Similarly, from the point of view of package P_j, we have $C^{\mathcal{I}_j} = r_{ij}(C^{\mathcal{I}_i}) = r_{ij}(\{x, y\}) = \{y\} \subseteq \{y, z\} = r_{ij}(\{y, z\}) = r_{ij}(D^{\mathcal{I}_i}) = D^{\mathcal{I}_j}$. Therefore, $\mathcal{I} \models_j C \sqsubseteq D$. However, in the partially overlapping domain semantics of [10], $C =_{\text{int}} D$ holds from both P_i's and P_j's point of view.

Thus, in spite of the fact that the intersection of two sets is "seen equally" from both sets' points of view, the example that was presented above illustrates that the way concept names are interpreted in these models still preserves some form of directionality in the subsumption reasoning.

Despite this subtle semantic difference between the partially overlapping domain semantics of [10] and the semantics of P-DL presented here, it is still possible to provide P-DL with a different kind of overlapping-domain-style semantics. More precisely, in the proof of Lemma 3, it is shown how one may combine the various local domains of a P-DL interpretation into one global domain. The P-DL model satisfies a given subsumption $C \sqsubseteq D$ from a witness P_i's point of view if and only if the global model satisfies an appropriately constructed *subjective* translation $\#_i(C) \sqsubseteq \#_i(D)$ of the given subsumption (see Section 3). Moreover, in the proof of Lemma 2, it is shown how, conversely, starting from a global domain, one may construct a P-DL model with various local domains; if the aforementioned subjective translation of a subsumption is satisfied in the global domain, then the original subsumption is satisfied from P_i's point of view. If the two constructions are composed, starting from the original P-DL model one obtains another equivalent model that is based on a partially-overlapping-style domain semantics. However, due to the interpretations of the translations of the concept names in this model, directionality is still preserved, unlike the situation in the ordinary partially overlapping domain semantics of [10].

Since any ordinary P-DL model gives rise to an equivalent model with partially-overlapping-style semantics, the question arises as to why the latter is not chosen as the fundamental notion of semantics for P-DL. The main reason is that, in many applications, local models are supposed to be populated independently of one another before semantic relations between their individuals are physically established. Moreover, the whole point of introducing modular description logics is to give temporally and spatially unrelated designers the chance to develop modules of a complex knowledge base independently. Additionally, the semantics of P-DL is derived from the Local Model Semantics [11], of which the directionality of domain relations, which will be lost in the partially-overlapping-style semantics, are crucial as domain relations also subjective. By keeping the directionality of domain relations, it also opens the possibility for various future extensions of P-DL when it is infeasible to use partially-overlapping-style semantics, e.g., when transitive knowledge propagation should be controlled among only trusted entities. □ (**End of Discussion**)

As immediate consequences of the proposed semantics for the P-DL $\mathcal{SHOIQP}$, extensions of various versions of the De Morgan's Law may be proven. Those deal with both the ordinary propositional logical connectives, including local negations, and with the quantifiers. For instance, it may be shown that, from the point of view of a package P_j which directly imports packages P_i and P_k, we have that $\neg_i(C \sqcap D) = \neg_i C \sqcup \neg_i D$ and also, $\neg_i(\forall R.C) = \neg_i \top_k \sqcup \exists R.\neg_j C$, where R is a k-role name. Similar semantic equivalences hold for various other connectives and quantifiers. Via these relations, proofs involving existential restriction and value restriction may be reduced to those involving the corresponding number restrictions.

In the next lemma, it is asserted that Condition 3 of Definition 1 holds not only for concept names, but, in fact, for arbitrary concepts. Beyond its own intrinsic interest, it becomes handy in Section 4 in showing that the package description logic $\mathcal{SHOIQP}$ supports monotonicity of reasoning and transitive reusability of modules.

Lemma 1. *Let Σ be a $\mathcal{SHOIQP}$ ontology, P_i, P_j two packages in Σ such that $P_i \in P_j^+$, C a concept such that* $\mathsf{Sig}(C) \subseteq \mathsf{Sig}(P_i) \cap \mathsf{Sig}(P_j)$*, and R a role name such that $R \in \mathsf{Sig}(P_i) \cap \mathsf{Sig}(P_j)$. If $\mathcal{I} = \langle \{\mathcal{I}_u\}, \{r_{uv}\}_{P_u \in P_v^+} \rangle$ is a model of Σ, then $r_{ij}(C^{\mathcal{I}_i}) = C^{\mathcal{I}_j}$ and $r_{ij}(R^{\mathcal{I}_i}) = R^{\mathcal{I}_j}$.*

The proof of Lemma 1 involves a structural induction on the concept formula C, that, by hypothesis, appears both in P_i and in P_j. The induction step employs the fact that, if $x' = r_{ij}(x)$, then

- $r_{ij} : R^{\mathcal{I}_i}(x) \to R^{\mathcal{I}_j}(x')$ is a total bijection and
- $r_{ij} : R^{\mathcal{I}_i}(x) \cap D^{\mathcal{I}_i} \to R^{\mathcal{I}_j}(x') \cap D^{\mathcal{I}_j}$ is also a total bijection, for every concept D, that appears in both P_i and P_j, and is such that $r_{ij}(D^{\mathcal{I}_i}) = D^{\mathcal{I}_j}$.

13.3 Reduction to Ordinary DL

In this section, we present a translation from concept formulas that appear in a given package of a $\mathcal{SHOIQP}$ KB Σ to concept formulas of a $\mathcal{SHOIQ}$ KB $\Sigma^\star$. The $\mathcal{SHOIQ}$ KB $\Sigma^\star$ is constructed in such a way that the top concept $\top_w$, associated with a specific package P_w of Σ, is satisfiable by $\Sigma^\star$ in the ordinary DL sense if and only if Σ itself is consistent *from the point of view of P_w* (see Theorem 1). (Note that the $\mathcal{SHOIQ}$ KB $\Sigma^\star$ is dependent on the importing relations present in $\mathcal{SHOIQP}$ Σ). This shows that the consistency problem in $\mathcal{SHOIQP}$ is reducible to the satisfiability problem in $\mathcal{SHOIQ}$, which is known to be NExpTime-complete [23, 24]. This has the consequence that the problems of concept satisfiability, concept subsumption and consistency in $\mathcal{SHOIQP}$ are also NExpTime-complete (see Theorem 2). Moreover, as will be seen in Section 4, this result also plays a central role in showing that some of the desiderata presented in Section 2.2 are satisfied by $\mathcal{SHOIQP}$. For instance, Reasoning Exactness, Monotonicity of Reasoning, Transitive Reusability of Knowledge and Preservation of Unsatisfiability are all features of $\mathcal{SHOIQP}$, which are shown to hold by employing the translation from $\mathcal{SHOIQP}$ to $\mathcal{SHOIQ}$, that will be presented in this section.

The reduction $\Re$ from a $\mathcal{SHOIQP}$ KB $\Sigma = \{P_i\}$ to a $\mathcal{SHOIQ}$ KB $\Sigma^\star$ can be obtained as follows: the signature of $\Sigma^\star$ is the union of the local signatures of the component packages together with a global top $\top$, a global bottom $\bot$ and local top concepts $\top_i$, for all i, i.e., $\mathsf{Sig}(\Sigma^\star) = \bigcup_i (\mathsf{Loc}(P_i) \cup \{\top_i\}) \cup \{\top, \bot\}$, and

a) For all i, j, k such that $P_i \in P_k^*$, $P_k \in P_j^*$, $\top_i \sqcap \top_j \sqsubseteq \top_k$ is added to $\Sigma^\star$.
b) For each GCI $X \sqsubseteq Y$ in P_j, $\#_j(X) \sqsubseteq \#_j(Y)$ is added to Σ. The mapping $\#_j()$ is defined below.
c) For each role inclusion $X \sqsubseteq Y$ in P_j, $X \sqsubseteq Y$ is added to $\Sigma^\star$.
d) For each i-concept name or i-nominal name C in P_i, $i : C \sqsubseteq \top_i$ is added to $\Sigma^\star$.
e) For each i-role name R in P_i, $\top_i$ is stipulated to be its domain and range, i.e., $\top \sqsubseteq \forall R^-.\top_i$ and $\top \sqsubseteq \forall R.\top_i$ are added to Σ.

f) For each i-role name R in P_j, the following axioms are added to $\Sigma^\star$:
   – $\exists R.\top_j \sqsubseteq \top_j$ (local domain);
   – $\exists R^-.\top_j \sqsubseteq \top_j$ (local range).
g) For each i-role name, add $\mathsf{Trans}(R)$ to $\Sigma^\star$ if $\mathsf{Trans}_i(R)$.

The mapping $\#_j()$ is adapted from a similar one for DDL [8] with modifications to facilitate context preservation whenever name importing occurs. For a formula X used in P_j, $\#_j(X)$ is:

- X, for a j-concept name or a j-nominal name.
- $X \sqcap \top_j$, for an i-concept name or an i-nominal name X.
- $\neg\#_j(Y) \sqcap \top_i \sqcap \top_j$, for $X = \neg_i Y$, where Y is a concept.
- $(\#_j(X_1) \oplus \#_j(X_2)) \sqcap \top_j$, for a concept $X = X_1 \oplus X_2$, where $\oplus = \sqcap$ or $\oplus = \sqcup$.
- $(\otimes R.\#_j(X')) \sqcap \top_i \sqcap \top_j$, for a concept $X = (\otimes R.X')$, where $\otimes \in \{\exists, \forall, \leq n, \geq n\}$ and R is an i-role.

For example, if C, D are concept names and R a role name,

$$\begin{aligned}
\#_j(\neg_i\, i : C) &= \neg(C \sqcap \top_j) \sqcap \top_i \sqcap \top_j \\
\#_j(j : D \sqcup i : C) &= (D \sqcup (C \sqcap \top_j)) \sqcap \top_j \\
\#_j(\forall(j : R).(i : C)) &= \forall R.(C \sqcap \top_j) \sqcap \top_j \\
\#_j(\exists(i : R).(i : C)) &= \exists R.(C \sqcap \top_j) \sqcap \top_i \sqcap \top_j
\end{aligned}$$

It should be noted that $\#_j()$ is *contextualized* so as to allow a given formula to have different interpretations when it appears in different packages. See also the Discussion subsection in Section 2.2.

13.4 Properties of Semantic Importing

In this section, we further justify the proposed semantics for $\mathcal{SHOIQP}$. More specifically, we present the main results showing that $\mathcal{SHOIQP}$ satisfies the desiderata listed in Section 2.

The first main theorem shows that the consistency problem of a $\mathcal{SHOIQP}$ ontology w.r.t. a witness package can be reduced to a satisfiability problem of a $\mathcal{SHOIQ}$ concept w.r.t. an integrated ontology *from the point of view of that witness package*, namely, $\Re(P_w^*)$. Note that there is no single universal integrated ontology for all packages. Each package, *sees* an integrated ontology (depending on the witness package and all the packages that are directly or indirectly imported by the witness package), and hence different packages can witness different consequences.

Theorem 1. *A $\mathcal{SHOIQP}$ KB Σ is consistent as witnessed by a package P_w if and only if $\top_w$ is satisfiable with respect to $\Re(P_w^*)$.*

Proof: Sufficiency is proven in Lemma 2 and necessity in Lemma 3. We present these two lemmas below, but give only outlines of their proofs. Detailed proofs are provided in [6].

Lemma 2. *Let Σ be a $\mathcal{SHOIQP}$ KB and P_w a package of Σ. If $\top_w$ is satisfiable with respect to $\Re(P_w^*)$, then Σ is consistent as witnessed by P_w.*

Proof: Assume that $\top_w$ is satisfiable with respect to $\Re(P_w^*)$ and let $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ be a model of $\Re(P_w^*)$, such that $\top_w^{\mathcal{I}} \neq \emptyset$. We construct a model $\langle \{\mathcal{I}_i\}, \{r_{ij}\}_{i \in P_j^*} \rangle$ of P_w^*, such that $\Delta^{\mathcal{I}_w} \neq \emptyset$. For each package $P_i \in P_w^*$, the local interpretation $\mathcal{I}_i$ is constructed as a projection of $\mathcal{I}$ in the following way:

- $\Delta^{\mathcal{I}_i} = \top_i^{\mathcal{I}}$;
- For every concept name C that appears in P_i, $C^{\mathcal{I}_i} = C^{\mathcal{I}} \cap \top_i^{\mathcal{I}}$;
- For every role name R that appears in P_i, $R^{\mathcal{I}_i} = R^{\mathcal{I}} \cap (\top_i^{\mathcal{I}} \times \top_i^{\mathcal{I}})$;
- For every nominal name o that appears in P_i, $o^{\mathcal{I}_i} = o^{\mathcal{I}}$;

and for every pair i, j, such that $P_i \in P_j^* \subseteq P_w^*$, we define

$$r_{ij} = \{(x, x) | x \in \Delta^{\mathcal{I}_i} \cap \Delta^{\mathcal{I}_j}\}.$$

Clearly, we have $\Delta^{\mathcal{I}_w} = \top_w^{\mathcal{I}} \neq \emptyset$, by the hypothesis. Moreover, it may be shown that $\langle \{\mathcal{I}_i\}, \{r_{ij}\}_{P_i \in P_j^*} \rangle$ is a model of the modular ontology P_w^*, i.e., that it satisfies the seven conditions postulated in Definition 1. The most challenging part is to show that, for every concept inclusion $C \sqsubseteq D$ in P_j, we must have $C^{\mathcal{I}_j} \subseteq D^{\mathcal{I}_j}$. Since, by the hypothesis, $\#_j(C)^{\mathcal{I}} \subseteq \#_j(D)^{\mathcal{I}}$ holds in $\mathcal{I}$, it suffices to show that, for every concept formula X that appears in P_j, we have $\#_j(X)^{\mathcal{I}} = X^{\mathcal{I}_j}$. This may be accomplished by structural induction on X. The details are omitted. Q.E.D.

Next, we proceed to show the reverse implication.

Lemma 3. *Let Σ be a $\mathcal{SHOIQP}$ KB. If Σ is consistent as witnessed by a package P_w, then $\top_w$ is satisfiable with respect to $\Re(P_w^*)$.*

Proof: Suppose that Σ is consistent as witnessed by P_w. Thus, it has a distributed model $\langle \{\mathcal{I}_i\}, \{r_{ij}\}_{P_i \in P_j^*} \rangle$, such that $\Delta^{\mathcal{I}_w} \neq \emptyset$. We proceed to construct a model $\mathcal{I}$ of $\Re(P_w^*)$ by merging individuals that are related via chains of image domain relations or their inverses. More precisely, for every element x in the distributed model, we define its equivalence class $\overline{x} = \{y | (x, y) \in \rho\}$ where ρ is the symmetric and transitive closure of the set $\bigcup_{P_i \in P_j^*} r_{ij}$. Moreover, for a set S, we define $\overline{S} = \{\bar{x} | x \in S\}$ and for a binary relation R, we define $\overline{R} = \{(\overline{x}, \overline{y}) | (x, y) \in R\}$.

A model $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ of Σ is now defined as follows:

- $\top^{\mathcal{I}} = \Delta^{\mathcal{I}} = \overline{\bigcup_i \Delta^{\mathcal{I}_i}}$, and $\bot^{\mathcal{I}} = \emptyset$.
- For every i-name X, $X^{\mathcal{I}} := \overline{X^{\mathcal{I}_i}}$.
- For every i, $\top_i^{\mathcal{I}} = \overline{\Delta^{\mathcal{I}_i}}$.

Next, it is shown that $\mathcal{I}$ is a model of $\Re(P_w^*)$, such that $\top_w^{\mathcal{I}} \neq \emptyset$. As in the proof of Lemma 2, the most challenging part is to show that, if $C \sqsubseteq D$ appears in P_j, then $\#_j(C)^{\mathcal{I}} \subseteq \#_j(D)^{\mathcal{I}}$ holds in $\mathcal{I}$. Since, by hypothesis, $C^{\mathcal{I}_j} \subseteq D^{\mathcal{I}_j}$ and this implies that $\overline{C^{\mathcal{I}_j}} \subseteq \overline{D^{\mathcal{I}_j}}$, it suffices to show that $\#_j(C)^{\mathcal{I}} = \overline{C^{\mathcal{I}_j}}$, for every concept formula C that appears in P_j. This is accomplished by induction on the structure of the concept C. The details can be found in [6]. Q.E.D.

Using Theorem 1 and the fact that concept satisfiability in $\mathcal{SHOIQ}$ is NExpTime-complete [23, 24], we obtain

Theorem 2. *The concept satisfiability, concept subsumption and consistency problems in* $\mathcal{SHOIQP}$ *are* NExpTime-*complete.*

The next theorem shows that concept subsumption problems in a $\mathcal{SHOIQP}$ ontology Σ, from the point of view of a specific witness package, can be reduced to concept subsumption problems in a *corresponding* $\mathcal{SHOIQ}$ ontology.

Theorem 3 (Reasoning Exactness). *For a* $\mathcal{SHOIQP}$ *KB* $\Sigma = \{P_i\}$, $C \sqsubseteq_j D$ *iff* $\Re(P_j^*) \models \#_j(C) \sqsubseteq \#_j(D)$.

Proof: As usual, we reduce subsumption to (un)satisfiability. It follows directly from Theorem 1 that P_j^* and $C \sqcap \neg_j D$ have a common model if and only if $\Re(P_j^*)$ and $\#_j(C) \sqcap \neg\#_j(D) \sqcap \top_j$ have a common model. Since $\#_j(C) \sqsubseteq \top_j$, this holds if and only if $\Re(P_j^*)$ and $\#_j(C) \sqcap \neg\#_j(D)$ have a common model. Thus, $\Re(P_j^*) \models \#_j(C) \sqsubseteq \#_j(D)$. Q.E.D.

Discussion of Desiderata

To show that the package description logic $\mathcal{SHOIQP}$ supports transitive reusability and preservation of unsatisfiability, we prove the monotonicity of reasoning in $\mathcal{SHOIQP}$.

Theorem 4 (Monotonicity and Transitive Reusability). *Suppose* $\Sigma = \{P_i\}$ *is a* $\mathcal{SHOIQP}$ *KB,* $P_i \in P_j^+$ *and* C, D *are concepts, such that* $\mathsf{Sig}(C) \cup \mathsf{Sig}(D) \subseteq \mathsf{Sig}(P_i) \cap \mathsf{Sig}(P_j)$. *If* $C \sqsubseteq_i D$, *then* $C \sqsubseteq_j D$.

Proof: Suppose that $C \sqsubseteq_i D$. Thus, for every model $\mathcal{I}$ of P_i^*, $C^{\mathcal{I}_i} \subseteq D^{\mathcal{I}_i}$. Now consider a model $\mathcal{J}$ of P_j^*. Since $P_i \in P_j^*$, $\mathcal{J}$ is also an interpretation of P_i^*. If $\bigcup_{P_k \in P_i^*} \Delta^{\mathcal{J}_k} = \emptyset$, then the conclusion holds trivially. Otherwise, $\mathcal{J}$ is a model of P_i^* and, therefore, $C^{\mathcal{J}_i} \subseteq D^{\mathcal{J}_i}$. Hence, $r_{ij}(C^{\mathcal{J}_i}) \subseteq r_{ij}(D^{\mathcal{J}_i})$, whence, by Lemma 1, $C^{\mathcal{J}_j} \subseteq D^{\mathcal{J}_j}$. This proves that $C \sqsubseteq_j D$. Q.E.D.

Theorem 4 ensures that when some part of an ontology module is reused, the restrictions asserted by it, e.g., domain restrictions on roles, will not be relaxed in a way that prohibits the reuse of imported knowledge. Theorem 4 also ensures that consequences of imported knowledge can be transitively propagated across importing chains.

In the special case where $D = \bot$, we obtain the following corollary:

Corollary 1 (Preservation of Unsatisfiability). *For a* $\mathcal{SHOIQP}$ *knowledge base* $\Sigma = \{P_i\}$ *and* $P_i \in P_j^+$, *if* $C \sqsubseteq_i \bot$ *then* $C \sqsubseteq_j \bot$.

Finally, the semantics of $\mathcal{SHOIQP}$ ensures that the interpretation of an axiom in an ontology module is constrained by its *context*, as seen from the reduction to a corresponding integrated ontology: $C \sqsubseteq D$ in P_j is mapped to $\#_j(C) \sqsubseteq \#_j(D)$, where $\#_j(C)$ and $\#_j(D)$ are now relativized to the corresponding local domain of P_j.

When a package P_i is directly or indirectly reused by another package P_j, some axioms in P_i may be effectively "propagated" to module P_j (i.e., may influence inference from the point of view of P_j). P-DL semantics ensures that such axiom propagation will affect only the "overlapping" domain $r_{ij}(\Delta^{\mathcal{I}_i}) \cap \Delta^{\mathcal{I}_j}$ and not the entire domain $\Delta^{\mathcal{I}_j}$.

Example 7. *For instance, in Figure 13.1, package P_1 contains an axiom $\neg_1\mathsf{Child} \sqsubseteq \mathsf{Adult}$ and package P_2 imports P_1. The assertion $\neg_1\mathsf{Child} \sqsubseteq \mathsf{Adult}$ is made within the implicit context of people, i.e. every individual that is not a child is an adult. Thus, every individual within the domain of people are either a Child or an Adult ($\top_1 \sqsubseteq \mathsf{Child} \sqcup \mathsf{Adult}$). However, it is not necessarily the case in P_2 that $\top_2 \sqsubseteq \mathsf{Child} \sqcup \mathsf{Adult}$. For example, an $\mathsf{Empolyer}$ in the domain of Work may be an organization which is not a member of the domain of People. In fact, since $r_{12}(\Delta^{\mathcal{I}_1}) \subseteq \Delta^{\mathcal{I}_2}$, $\Delta^{\mathcal{I}_1} \backslash \mathsf{Child}^{\mathcal{I}_1} \subseteq \mathsf{Adult}^{\mathcal{I}_1}$, i.e., $\Delta^{\mathcal{I}_1} = \mathsf{Child}^{\mathcal{I}_1} \cup \mathsf{Adult}^{\mathcal{I}_1}$, does not necessarily imply $\Delta^{\mathcal{I}_2} = \mathsf{Child}^{\mathcal{I}_2} \cup \mathsf{Adult}^{\mathcal{I}_2}$.*

Hence, the effect of an axiom is always limited to its original designated context. Consequently, it is not necessary to explicitly restrict the use of the ontology language to ensure locality of axioms, as is required, for instance, by conservative extensions [13]. Instead, the locality of axioms follows directly from the semantics of $\mathcal{SHOIQP}$.

13.5 Discussion of the P-DL Semantics

13.5.1 Necessity of P-DL Constraints on Domain Relations

The constraints on domain relations in the semantics of $\mathcal{SHOIQP}$, as given in Definition 1, are minimal in the sense that if we drop any of them, we can no longer satisfy the desiderata summarized in Section 13.2.2.

Dropping Condition 1 of Definition 1 (one-to-one domain relations) leads to difficulties in preservation of concept unsatisfiability. For example, if the domain relations are not injective, then $C_1 \sqsubseteq_i \neg_i C_2$, i.e., $C_1 \sqcap C_2 \sqsubseteq_i \bot$, does not ensure $C_1 \sqcap C_2 \sqsubseteq_j \bot$ when P_j imports P_i. If the domain relations are not partial functions, multiple individuals in $\Delta^{\mathcal{I}_j}$ may be images of the same individual in $\Delta^{\mathcal{I}_i}$ via r_{ij}, whence unsatisfiability of a complex concept can no longer be preserved when both number restriction and role importing are allowed. Thus, if R is an i-role name and C is an i-concept name, $\geq 2R.C \sqsubseteq_i \bot$ does not imply $\geq 2R.C \sqsubseteq_j \bot$.

Dropping Condition 2 of Definition 1 (compositional consistency of domain relations) would result in violation of the transitive reusability requirement, in particular, and of the monotonicity of inference based on imported knowledge, in general. In the absence of compositional consistency of domain relations, the importing relations would be like bridge rules in DDL, in that they are localized w.r.t. the connected pairs of modules without supporting compositionality [25].

In the absence of Conditions 3 and 4 of Definition 1, the reuse of concept and role names would be purely syntactical, i.e., the local interpretations of imported concepts and role names would be unconstrained by their interpretations in their home package.

Condition 5 (cardinality preservation of role instances) is needed to ensure the consistency of local interpretations of complex concepts that use number restrictions.

Condition 6 is needed to ensure that concepts that are nominals can only have one instance. Multiple "copies" of such an instance are effectively identified with a single instance via domain relations.

Finally, Condition 7, i.e., that $\mathcal{I}_i \models P_i$, for every i, is self-explanatory.

13.5.2 Contextualized Negation

Contextualized negation has been studied in logic programming [21, 22]. Existing modular ontology languages DDL and $\mathcal{E}$-Connections do not explicitly support contextualized negation in their respective syntax. In fact, in those formalisms, a negation is always interpreted with respect to the local domain of the module in which the negation occurs, not the union of all local domains. Thus, in fact, both DDL and $\mathcal{E}$-Connections implicitly support contextualized negation.

The P-DL syntax and semantics, proposed in this work, support a more general use of contextualized negation so that a package can use, besides its own negation, the negations of its imported packages[5].

13.5.3 Directionality of Importing

There appears to be some apparent confusion in the literature regarding whether the constraints imposed by P-DL allow the importing relations in P-DL to be indeed directional [15]. As noted by Grau [15], if it is indeed the case that a P-DL model $\mathcal{I}$ satisfies $r_{ij}(s^{\mathcal{I}_i}) = s^{\mathcal{I}_j}$ if only if it satisfies $r_{ji}(s^{\mathcal{I}_j}) = s^{\mathcal{I}_i}$, for any symbol s such that $P_i \xrightarrow{s} P_j$ (Definition 18 and Proposition 19 in [15]) it must follow that a P-DL ontology can be reduced to an equivalent imports-free ontology. Then, a shared symbol s of P_i and P_j must have the same interpretation from the point of view of both P_i and P_j, i.e., $s^{\mathcal{I}_i} = s^{\mathcal{I}_j}$. However, according to our definition of model (Definition 1), it is *not* the case that a P-DL model $\mathcal{I}$ satisfies $r_{ij}(s^{\mathcal{I}_i}) = s^{\mathcal{I}_j}$ if only if it satisfies $r_{ji}(s^{\mathcal{I}_j}) = s^{\mathcal{I}_i}$, for any symbol s such that $P_i \xrightarrow{s} P_j$. As noted by Bao et al. [2, 3]:

- P-DL semantics does not require the existence of both r_{ij} and r_{ji}. Their joint existence is only required when P_i and P_j mutually import one another. Hence, even if $r_{ij}(s^{\mathcal{I}_i}) = s^{\mathcal{I}_j}$, it is possible that the corresponding r_{ji} may not exist in which case $r_{ji}(s^{\mathcal{I}_j})$ is undefined.

[5] We thank Jeff Pan for discussions on this issue.

- Domain relations are *not necessarily* total functions. Hence, it need not be the case that *every* individual of $\Delta^{\mathcal{I}_i}$ is mapped (by the one-to-one domain relation r_{ij}) to an individual of $\Delta^{\mathcal{I}_j}$.
- Satisfiability and consistency have only contextualized meaning in P-DL. If P_j is not in P_i^*, then models of P_i^* need not be models of P_j^*. This is made clear in Definition 2 where satisfiability and consistency are always considered from the point of view of a witness package.

In the following subsection, we will present an additional example (Example 8) that illustrates the directionality of importing in P-DL.

13.5.4 P-DL Consistency and TBox Consistency

In Section 13.3 we have shown how to reduce a $\mathcal{SHOIQP}$ P-DL ontology to a corresponding DL ($\mathcal{SHOIQ}$) ontology. We have further shown (Theorem 1) that determining the consistency of a $\mathcal{SHOIQP}$ ontology from the point of view of a package P_w can be reduced to the satisfiability of a $\mathcal{SHOIQ}$ concept with respect to a $\mathcal{SHOIQ}$ ontology obtained by integrating the packages imported by P_w. However, it is important to note that this reduction of $\mathcal{SHOIQP}$ is *different* from a reduction based on S-compatibility as defined in [15].

Definition 3 (Expansion). [15] *Let A-interpretation denote an interpretation over a signature A. An S-interpretation $\mathcal{J} = (\Delta^{\mathcal{J}}, \cdot^{\mathcal{J}})$ is an expansion of an S'-interpretation $\mathcal{J}' = (\Delta^{\mathcal{J}'}, \cdot^{\mathcal{J}'})$ if*

(1) $S' \subseteq S$,
(2) $\Delta^{\mathcal{J}'} \subseteq \Delta^{\mathcal{J}}$, *and*
(3) $s^{\mathcal{J}} = s^{\mathcal{J}'}$, *for every* $s \in S'$.

Definition 4 (S-compatibility). [15] *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be TBoxes expressed in a description logic $\mathcal{L}$, and let S be the shared part of their signatures. We say that $\mathcal{T}_1$ and $\mathcal{T}_2$ are S-compatible if there exists an S-interpretation $\mathcal{J}$, that can be expanded to a model $\mathcal{J}_1$ of $\mathcal{T}_1$ and to a model $\mathcal{J}_2$ of $\mathcal{T}_2$.*

As the following example illustrates, a P-DL ontology is not always reducible to the imports-free ontology that is obtained by simply taking the union of the modules (packages).

Example 8. *Let $\mathcal{T}_1 = \{D \sqcup \neg D \sqsubseteq C\}$, $\mathcal{T}_2 = \{C \sqsubseteq \bot\}$. The shared signature $S = \{C\}$ and $\mathcal{T}_1$ and $\mathcal{T}_2$ are not S-compatible. However, suppose we have a* P-DL *ontology such that $\mathcal{T}_1 \xrightarrow{C} \mathcal{T}_2$ and negation in $\mathcal{T}_1$ becomes contextualized negation $\neg_1$. Then we have a model:*

$$\Delta_1 = C^{\mathcal{I}_1} = D^{\mathcal{I}_1} = \{x\}$$
$$\Delta_2 = \{y\}, C^{\mathcal{I}_2} = \emptyset$$
$$r_{12} = r_{21} = \emptyset$$

On the other hand, all models of a P-DL *ontology where $\mathcal{T}_2 \xrightarrow{C} \mathcal{T}_1$ have empty Δ_1. Thus, the whole ontology is consistent as witnessed by $\mathcal{T}_2$ but inconsistent as witnessed by $\mathcal{T}_1$. This example demonstrates that* P-DL *importing is directional.*

The next example shows that, in the presence of nominals, the P-DL consistency problem is not reducible to the consistency of an imports-free ontology obtained by simply combining the P-DL modules.

Example 9 (Use of Nominals). *Consider the following TBoxes:*

$$\mathcal{T}_1 = \{\top \sqsubseteq i \sqcup j, \quad i \sqcap j \sqsubseteq \bot\}$$
$$\mathcal{T}_2 = \{\top \sqsubseteq i\},$$

with the shared signature $S = \{i\}$, where i, j are nominals. $\mathcal{T}_1$ and $\mathcal{T}_2$ are S-compatible but $\mathcal{T}_1 \cup \mathcal{T}_2$ is not consistent. Suppose we have a P-DL *ontology with $\mathcal{T}_1 \xrightarrow{i} \mathcal{T}_2$. Since "$\top$" only has contextualized meaning in* P-DL, *these TBoxes in fact should be represented as*

$$\mathcal{T}_1 = \{\top_1 \sqsubseteq i \sqcup j, \quad i \sqcap j \sqsubseteq \bot\}$$
$$\mathcal{T}_2 = \{\top_2 \sqsubseteq i\}$$

Now, there exists a model for this P-DL *ontology:*

$$\Delta_1 = \{x, y\}, i^{\mathcal{I}_1} = \{x\}, j^{\mathcal{I}_1} = \{y\}$$
$$\Delta_2 = \{x'\}, i^{\mathcal{I}_2} = \{x'\}$$
$$r_{12} = \{(x, x')\}$$

In general, the reduction from P-DL modules to imports-free TBoxes with shared signatures based on S-compatibility, as suggested by [15], does not preserve the semantics of P-DL. Thus, there is a fundamental difference between the two settings: P-DL has no universal top concept and, as a result, P-DL axioms have only localized effect. In the case of imports-free TBoxes, in the absence of contextualized semantics, it is not possible to ensure that the effects of axioms are localized. Consequently, it is not possible to reduce reasoning with a P-DL ontology with modules $\{\mathcal{T}_i\}$ to standard DL reasoning over the union of all ontology modules $\mathcal{T} = \mathcal{T}_1 \cup ... \cup \mathcal{T}_n$.

In contrast, in the previous section we have shown that such a reduction from reasoning in P-DL from the point of view of a witness package to reasoning with a suitably constructed DL (as shown in Section 13.3) is possible. Nevertheless, relying on such a reduction is not attractive in practice, because it requires the integration of the ontology modules, which may be prohibitively expensive. More importantly, in many scenarios encountered in practice, e.g., in peer-to-peer applications, centralized reasoning with an integrated ontology is simply infeasible. Hence, work in progress is aimed at developing federated reasoners for P-DL that do not require the integration of different ontology modules (see, e.g., [4]).

13.6 Summary

In this chapter, we have introduced a modular ontology language, package-based description logic $\mathcal{SHOIQP}$, that allows reuse of knowledge from multiple ontologies. A $\mathcal{SHOIQP}$ ontology consists of multiple ontology modules each of which can be viewed as a $\mathcal{SHOIQ}$ ontology. Concept, role and nominal names can be shared by "importing" relations among modules.

The proposed language supports contextualized interpretation, i.e., interpretation from the point of view of a specific package. We have established a minimal set of constraints on domain relations, i.e., the relations between individuals in different local domains, that allow the preservation of the satisfiability of concept expressions, the monotonicity of inference, and the transitive reuse of knowledge.

Ongoing work is aimed at developing a distributed reasoning algorithm for $\mathcal{SHOIQP}$ by extending the results of [4] and [20], as well as an OWL extension capturing the syntax of $\mathcal{SHOIQP}$. We are also exploring several variants of P-DL, based on a more in-depth analysis of the properties of the domain relations and the preservation of satisfiability of concept subsumptions across modules.

Acknowledgement

This work was supported in part by the grants IIS 0639230 and IIS 0711356 from the National Science Foundation.

References

1. Bao, J., Caragea, D., Honavar, V.: A distributed tableau algorithm for package-based description logics. In: The 2nd International Workshop On Context Representation And Reasoning (CRR 2006), co-located with ECAI 2006 (2006)
2. Bao, J., Caragea, D., Honavar, V.: Modular ontologies - a formal investigation of semantics and expressivity. In: Mizoguchi, R., Shi, Z.-Z., Giunchiglia, F. (eds.) ASWC 2006. LNCS, vol. 4185, pp. 616–631. Springer, Heidelberg (2006)
3. Bao, J., Caragea, D., Honavar, V.: On the semantics of linking and importing in modular ontologies. In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L.M. (eds.) ISWC 2006. LNCS, vol. 4273, pp. 72–86. Springer, Heidelberg (2006)
4. Bao, J., Caragea, D., Honavar, V.: A tableau-based federated reasoning algorithm for modular ontologies. In: IEEE/WIC/ACM International Conference on Web Intelligence, pp. 404–410. IEEE Press, Los Alamitos (2006)
5. Bao, J., Slutzki, G., Honavar, V.: A semantic importing approach to knowledge reuse from multiple ontologies. In: AAAI, pp. 1304–1309 (2007)
6. Bao, J., Voutsadakis, G., Slutzki, G., Honavar, V.: Package based description logics. Technical report, TR 567, Computer Science, Iowa State University (2008), `http://archives.cs.iastate.edu/documents/disk0/00/00/05/67/index.html`

7. Berners-Lee, T., Hendler, J., Lassila, O.: The semantic web. Scientific American 284(5), 34–43 (2001)
8. Borgida, A., Serafini, L.: Distributed description logics: Assimilating information from peer sources. Journal of Data Semantics 1, 153–184 (2003)
9. Bouquet, P., Giunchiglia, F., van Harmelen, F., Serafini, L., Stuckenschmidt, H.: C-OWL: Contextualizing ontologies. In: Fensel, D., Sycara, K.P., Mylopoulos, J. (eds.) ISWC 2003. LNCS, vol. 2870, pp. 164–179. Springer, Heidelberg (2003)
10. Catarci, T., Lenzerini, M.: Representing and using interschema knowledge in cooperative information systems. In: CoopIS, pp. 55–62 (1993)
11. Ghidini, C., Giunchiglia, F.: Local models semantics, or contextual reasoning=locality+compatibility. Artificial Intelligence 127(2), 221–259 (2001)
12. Ghilardi, S., Lutz, C., Wolter, F.: Did i damage my ontology? a case for conservative extensions in description logics. In: KR, pp. 187–197 (2006)
13. Grau, B.C., Horrocks, I., Kazakov, Y., Sattler, U.: A logical framework for modularity of ontologies. In: IJCAI, pp. 298–303 (2007)
14. Grau, B.C., Horrocks, I., Kutz, O., Sattler, U.: Will my ontologies fit together? In: Proc. of the 2006 Description Logic Workshop (DL 2006), vol. 189. CEUR (2006), `http://ceur-ws.org/`
15. Grau, B.C., Kutz, O.: Modular ontology languages revisited. In: Workshop on Semantic Web for Collaborative Knowledge Acquisition (SWeCKa), co-located with IJCAI (2007)
16. Grau, B.C., Parsia, B., Sirin, E.: Working with multiple ontologies on the semantic web. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 620–634. Springer, Heidelberg (2004)
17. Horrocks, I., Sattler, U.: A Tableaux Decision Procedure for $\mathcal{SHOIQ}$. In: IJCAI, pp. 448–453 (2005)
18. Horrocks, I., Sattler, U., Tobies, S.: Practical reasoning for expressive description logics. In: Ganzinger, H., McAllester, D., Voronkov, A. (eds.) LPAR 1999. LNCS, vol. 1705, pp. 161–180. Springer, Heidelberg (1999)
19. Lutz, C., Walther, D., Wolter, F.: Conservative extensions in expressive description logics. In: IJCAI, pp. 453–458 (2007)
20. Pan, J., Serafini, L., Zhao, Y.: Semantic import: An approach for partial ontology reuse. In: 1st International Workshop on Modular Ontologies (WoMo 2006), co-located with ISWC (2006)
21. Polleres, A.: Logic programs with contextually scoped negation. In: WLP, pp. 129–136 (2006)
22. Polleres, A., Feier, C., Harth, A.: Rules with contextually scoped negation. In: Sure, Y., Domingue, J. (eds.) ESWC 2006. LNCS, vol. 4011, pp. 332–347. Springer, Heidelberg (2006)
23. Tobies, S.: The complexity of reasoning with cardinality restrictions and nominals in expressive description logics. J. Artif. Intell. Res. (JAIR) 12, 199–217 (2000)
24. Tobies, S.: Complexity results and practical algorithms for logics in Knowledge Representation. Phd thesis, LuFG Theoretical Computer Science, RWTH-Aachen, Germany (2001)
25. Zimmermann, A., Euzenat, J.: Three semantics for distributed systems and their relations with alignment composition. In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L.M. (eds.) ISWC 2006. LNCS, vol. 4273, pp. 16–29. Springer, Heidelberg (2006)

Author Index

Bao, Jie 349
Borgida, Alexander 91
Brockmans, Saartje 267

Cuenca Grau, Bernardo 159, 293

d'Aquin, Mathieu 67

Haase, Peter 267
Honavar, Vasant 349
Horrocks, Ian 159

Kazakov, Yevgeny 159
Konev, Boris 25

Lutz, Carsten 25

Musen, Mark A. 245

Noy, Natalya F. 245

Parent, Christine 5, 113
Parsia, Bijan 293

Sabou, Marta 67
Sattler, Ulrike 159
Schlicht, Anne 67, 187
Seidenberg, Julian 211
Serafini, Luciano 267, 321
Sirin, Evren 293
Slutzki, Giora 349
Spaccapietra, Stefano 5, 113
Stuckenschmidt, Heiner 67, 187, 267

Tamilin, Andrei 321

Voutsadakis, George 349

Walther, Dirk 25
Wolter, Frank 25

Zimányi, Esteban 113

Subject Index

$\mathcal{ALC}$ 28, 161
$\mathcal{ALCO}$ 162
$\mathcal{EL}$ 161
$\mathcal{SHIF}(\mathcal{D})$ 293
$\mathcal{SHIQ}$ 162
$\mathcal{SHOIN}(\mathcal{D})$ 293
$\mathcal{SHOIQ}$ 162
$\mathcal{SHOIQP}$ 352
  reduction to description logics, 362
  semantics, 355
  Syntax, 352

A-box 44
abstract description system 295
abstract syntax 306
acyclic ontology 55
annotation 306
anonymous class 214
arrow variable 272
assertion 161
atomic concept 160
atomic role 161
attribute
  perception-varying, 128
  space-varying, 123
  time-varying, 123
automatic configuration 197
axiom 161, 300
axiom duplication 196

backward link 214
boundary class 220
boundary extraction 230
bridge graph 325
bridge operator 329
bridge rule 324
  equivalence, 324
  into, 324
  onto, 324
  satisfiability, 327
bulkyness 195

C-OWL 276
class 295
cohesion 73
combined interpretation 309
completeness 72, 253
completion graph 315
complexity 8
  computational, 38
composition 16
conceptual model 116
connectedness 74, 195
conservative extension 33, 165
constraining relationship 123
context 9, 20, 148, 350
contextualized negation 367
correctness 15, 72
coverage 74
criteria 71, 194
Cyc microtheories 148
Cyc ontology 95
  microtheories, 115

datatype 295
datatype map 308
DDL *see* distributed description logics

syntax, 323
decidability 166
decision problem 35
defined classes 213
definitorial extension 35
dependency
between concepts, 187
internal, 192
dependency graph 188
depth limiting 219
Description Logics 160
description logics 295
distributed, 323
package-based, 349
directionality 367
disjointness 11
distributed description logics 276, 323
inference services, 327
semantics, 325
distributed first-order logic 272
satisfiability, 274
distributed inference services 327
DL-lite 56
domain relation 273, 326
DRAGO 335

$\mathcal{E}$-Connections 105, 279, 294
encapsulation 71
essential axiom 165
Evaluation 72, 84
expanded vocabulary 100
expressive power 107
of description logics, 107
of local model-based logics, 109

FaCT 313
fact 300
FaCT++ 214
first-order logic 30
FMA ontology 212
foreign entity 298
forgetting 52

GALEN ontology 184, 211
Gene ontology 184
Google 212
granularity 195

hierarchy 216
hole interpretation 326

import
of modules, 102
inconsistency 39
independence 72
individual 161
information hiding 71
inseparability 33, 46
instance 295
Inter-module distance 75
inter-module links 18
interperception link 125, 126, 130–132
interpolation 48
interpretation 162
distributed-, 327
interpretation constraint 274
intra-module distance 73

KMi 83
knowledge import 92
knowledge propagation 328
knowledge selection 76

lightweight description logics 55
line island 190
height of a, 192
link
is-a, 118
local, 126
overlap, 118
symmetric, 216
link property 295
local domains 326
local interpretation 325
local interpretation domain 326
locality 173
locality-based module 175
local model semantics 272, 325

machine learning 14
Magpie 76
mapping
UML profile of a, 287
mapping axiom 274
mapping language 267
assumptions, 271
graphical modelling, 285

metamodel 268, 282
  of mappings, 282
  of OWL, 282
  of queries, 284
minimal module 165
model 162
modularity 294
modularization
  semantics-based, 101
  semantics-driven, 12
  structure-based, 13, 188
  syntactic, 100
modular ontology 327
module 159, 187, 294
module extraction 70, 103
Module relations 74
module size 73
multi-instantiation 118
multiperception database 115
  creation of, 126–127
multiple representations
  in DBMS and GIS, 114

NCI ontology 184, 212
Normalisation 213

ontolingua 93
ontology 159
ontology evolution 255
ontology extraction 220
  by traversal, 238
ontology integration framework 277
ontology interconnection 150
ontology mapping 270
ontology partitioning 70
ontology segmentation 70, 212
ontology traversal 71
optimization 197
overlap 11
OWL *see* Web Ontology Language
  metamodel, 282
  UML profile of, 285
owl:imports 294
OWL API 183, 304
OWL DL 160, 293

package-based description logics 349
partitioning 237
  structure-based, 237
PATO 81, 197
PDL *see* package-based description logics
Pellet 214, 295, 313
perception
  composite, 124, 125
  context vs., 148
  defined, 115
  dependencies, 135–137
  implementation, 142
  implemention, 146
  simple, 124, 125
  use of a, 137–142
  versioning vs., 147–148
  view vs., 147
Performance 76
poly-hierarchy 213
primitive classes 213
probe class 229
PROMPT 239
  PROMPT-FACTOR, 183
Prompt 82
property 213, 295
  filtering, 218
  meta-, 218
property hierarchy 217

quality 73
query
  conjunctive, 44
query language 107

RACER 214, 313
reasoner 313
  tableaux-based, 216
reasoning
  distributed-, 331
  tableau-, 315
reasoning peer 335
reciprocal link 214
Redundancy 74
representation 113–116, 125–129
restriction 214
reuse 9
richness 74
Robinson joint consistency 48
robustness 36, 41

satisfiability
  distributed-, 327

scalability 7
segmentation
  -algorithm, 217
semantic import 352
semantic relation 270
semantic web 212
SEP triples 223
signature 160
SNOMED-CT 212
social network theory 189
space and time
  continuous (field) view, 123
  discrete (object) view, 121
SPARQL 235
spatial data type 122
subschema 115
subsumption propagation 328, 329
SUMO ontology 184
surgical procedures 211
SWOOP 80
Swoop 295, 305

T-box
  distributed-, 324
tableau
  distributed, 332
temporal data type 122
topology 214
transitive propagation 221
traversal directive 249
  composition, 252
trivially equivalent definition 219

UML profile
  of a mapping, 287
  OWL, 285
uniform interpolation 52
URI *see* Uniform Resource Identifier
Use cases 68
User interaction 75

variable elimination 52
version 147–148
view 114–115, 147
  ontology-, 246
  traversal-, 249
view extraction 70
visualization
  of criteria dependencies, 200
  of large ontologies, 201

Web Ontology Language 212
WordNet 94