Advances in Intelligent Systems and Computing 626

Grzegorz J. Nalepa
Joachim Baumeister *Editors*

Synergies Between Knowledge Engineering and Software Engineering

Springer

Advances in Intelligent Systems and Computing

Volume 626

Series editor

Janusz Kacprzyk, Polish Academy of Sciences, Warsaw, Poland
e-mail: kacprzyk@ibspan.waw.pl

About this Series

The series "Advances in Intelligent Systems and Computing" contains publications on theory, applications, and design methods of Intelligent Systems and Intelligent Computing. Virtually all disciplines such as engineering, natural sciences, computer and information science, ICT, economics, business, e-commerce, environment, healthcare, life science are covered. The list of topics spans all the areas of modern intelligent systems and computing.

The publications within "Advances in Intelligent Systems and Computing" are primarily textbooks and proceedings of important conferences, symposia and congresses. They cover significant recent developments in the field, both of a foundational and applicable character. An important characteristic feature of the series is the short publication time and world-wide distribution. This permits a rapid and broad dissemination of research results.

More information about this series at http://www.springer.com/series/11156

Grzegorz J. Nalepa · Joachim Baumeister
Editors

Synergies Between Knowledge Engineering and Software Engineering

Springer

Editors

Grzegorz J. Nalepa
AGH University of
Science and Technology
Kraków
Poland

Joachim Baumeister
Universität Würzburg
Würzburg
Germany

ISSN 2194-5357 ISSN 2194-5365 (electronic)
Advances in Intelligent Systems and Computing
ISBN 978-3-319-64160-7 ISBN 978-3-319-64161-4 (eBook)
https://doi.org/10.1007/978-3-319-64161-4

Library of Congress Control Number: 2017947445

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer International Publishing AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

To all the people involved in ten editions of the KESE workshop

Preface

In the early beginnings, the construction of automated programs for computers was seen as a special kind of scientific art. With the emergent use and the advent of higher-level programming languages, the term "Software Engineering" was coined in the late 1970s as a title of a NATO conference bringing together international experts in that field. At that conference, the experts wanted to tackle the pressing problems of software programming, resulting in poor stability and exceeding costs of the software. They aimed for a complete redefinition, where the development of software was seen as "engineering discipline". A systematic and measurable approach of building computer programs was the ultimate goal. Following the principles of established engineering disciplines, the quality of developing software should then also improve. As of today, Software Engineering (SE) distinguishes a number of areas, related to the software life cycle, most importantly requirements engineering, software design, quality management (including testing), construction, and maintenance; and furthermore, software development methodologies and programming languages.

With improved methods and practices, the quality of software clearly improved over the decades, despite the dramatically increased complexity. Also the development costs decreased by order of magnitudes when compared to the complexity of the software artifacts. However, as Fred Brooks prominently said, there will be no "silver bullet" for building software in a perfect and error-less manner.

The development of the first larger expert systems started around the 1970s as special purpose computer programs. In the early years, the development process was similar to the early software engineering days: The construction of expert systems followed no systematic approach, often specialized inference engines were programmed ad-hoc in hand with the knowledge acquisition phase. Custom tools were developed from project to project. In consequence, the same problems popped-up as well-known from the early software engineering days: Projects developed unpredictable costs, the quality of the systems was not measurable, and many systems were abandoned due to their non-maintainability. The term "Knowledge Engineering" (KE) was born to tackle these problems. With the same systematic approach, many established methods and practices from Software

Engineering were adapted to the needs of building intelligent systems. With the development of tailored knowledge-based methods, the Software Engineering community realized the potential of intelligent methods for Software Engineering and thus the intellectual loop was closed by influential methods of Knowledge Engineering in Software Engineering. Until today, this symbiotic relationship exists and produces fruitful results. Currently, a number of international journals and conferences are devoted to the interrelations of Software Engineering and Knowledge Engineering.

In recent years, this relation is even stronger, with the rise of "intelligent/smart" software penetrating many aspects of daily lives. Smart homes, intelligent assistants, chatbots, autonomous cars, and intelligent manufacturing; these all are software systems that require knowledge to develop and employ their intelligent behavior and operation to the end user.

This volume compiles a number of submissions originated in the KESE workshop series: The first workshop *KESE: Knowledge Engineering and Software Engineering* was held in 2005 at the 28th German Conference on Artificial Intelligence (KI-2005) in Koblenz, Germany. The idea of the organizers was the realignment of the discipline Knowledge Engineering and its strong relation to Software Engineering, as well as to the classical Artificial Intelligence (AI) research. The practical aspects of AI systems emphasize the need for combining KE and SE methods and techniques. Due to their initial success, the KESE workshops were annually organized at the German AI conference (KI). In the years 2006–2010 KESE was held together with the KI in Bremen, Osnabruck, Kaiserslautern, Paderborn, and Karlsruhe. In these years, the workshop gathered a motivated and active international community. Thanks to it, in 2011 the KESE7 workshop was co-located with the Spanish AI conference CAEPIA 2011, held in San Cristobal de la Laguna, Tenerife. Then, in 2012 KESE moved the European AI conference, ECAI, then held in Montpellier. In 2013, it shortly moved back to the KI, held in Koblenz again. The tenth and last edition of KESE occurred in 2014. It was again co-located with ECAI then held in Prague. In its ten years of continuous existence, KESE attracted a stable flow of high quality papers. In the years 2005–2014, the following numbers of papers were presented at KESE: 8, 5, 6, 9, 7, 5, 7, 10, 8, 10, total of 75 papers. Starting from 2007, the workshop had its own proceedings published via CEUR WS website (http://ceur-ws.org), as CEUR volumes: 282, 425, 486, 636, 805, 949, 1070, and 1289. Moreover, in 2011 a Special Issue on Knowledge and Software Engineering for Intelligent Systems of the International Journal of Knowledge Engineering and Data Mining Vol. 1 No. 3 was published. All the details of the history of KESE were made available on a dedicated website: http://kese.ia.agh.edu.pl

The KESE community not only appreciated the scientific quality of the workshop, but also its important social aspects. Every year the participants held additional vibrant discussions during the so-called balcony-sessions, commonly referred to as b-sessions.

The workshop series always encouraged submissions describing methodological research combining Knowledge Engineering and Software Engineering but also the presentation of successful applications demanding for both disciplines. In fact, besides regular papers, starting from 2009 KESE also solicited tool presentations. In the 10 years of annual workshops, the topics of interest varied from AI methods in software/knowledge engineering (knowledge and experience management, declarative, logic-based approaches, constraint programming, agent-oriented software engineering, issues of maintenance) and knowledge/software engineering methods in AI (collaborative engineering of the Semantic Web, database and knowledge base management in AI systems, tools for intelligent systems, evaluation of (intelligent) systems: verification, validation, assessment, process models) to a range of topics located on the intersection of several disciplines relevant for KESE. In the final edition these included: knowledge and software engineering for the Semantic Web, knowledge and software engineering for Linked Data, ontologies in practical knowledge and software engineering, business systems modeling, design and analysis using KE and SE, practical knowledge representation and discovery techniques in software engineering, context and explanation in intelligent systems, knowledge base management in KE systems, evaluation and verification of KBS, practical tools for KBS engineering, process models in KE applications, software requirements and design for KBS applications, software quality assessment through formal KE models, and declarative, logic-based, including constraint programming approaches in SE.

After 10 bright years of annual workshops, we decided it is the time summarize the decade of works of the KESE community. Thus, we decided to prepare and edit this "memorial" volume. It compiles thirteen intriguing contributions closely related to the KESE topics with both methodological and application background grouped in two separate parts.

The first methodological part is composed of seven chapters. Ralph Schäfermeier and Adrian Paschke introduce an ontology development process that is inspired by Aspect-Oriented Programming. By following the aspect-oriented development, the complexity of the construction process can be simplified by strict modularization. Ralph Bergmann and Gilbert Müller improve the (re-)use of workflows in process-oriented information systems by introducing methods for the retrieval and adaptation of (best practice) workflows applicable to new process problems. Isabel María Del Águila and José Del Sagrado describe an approach to building of intelligent systems using Bayesian networks. They use UML known from Software Engineering and introduce the meta-model BayNet that supports the development and maintenance process. How can software development be supported by Knowledge Engineering techniques? Paraskevi Smiari et al. represent anti-patterns in software development by Bayesian networks and include this into a knowledge-based framework. During the software development the data is acquired and problems are identified as anti-patterns. Bayesian networks are used to assess the anti-pattern in the concrete problem situation. The quality of intelligent systems is considered by the contribution of Rainer Knauf: He introduces formal methods for the validation and finally the refinement of intelligent systems. Automated methods for test case generation and support for the human inspection are presented. The quality of intelligent systems is also discussed in the next contribution:

Marius Brezovan and Costin Badica present a novel method for the verification of knowledge-based systems. They apply a mathematical language for modeling the static and dynamic properties of the systems. Finally, in her contribution, Kerstin Bach tackles the problem of building large (knowledge-intensive decision-support) systems. With the industry-inspired term "Knowledge Line" a methodology for the modularization of knowledge is presented and exemplified.

The second part is devoted to applications and is composed of six chapters. The authors Paolo Ciancarini et al. report on the development of a mission-critical and knowledge-intensive system. They define the process model iAgile which is based on the successful Scrum methodology widely applied in Software Engineering. Pascal Reuss et al. also report on construction of an intelligent system, here the diagnosis and maintenance of civil aircrafts. A multi-agent approach is used to organize and implement the problem domain. A practical application report from the medical domain is given by Paulo Novais et al. A model of computer interpretable guidelines is introduced by using the ontology language OWL. Together with the language, an implementation in a clinical system is given. The web traversals and actions of users in the web is an interesting asset for market research. Adrian Giurca introduces the ontology Metamarket that models the user preferences and interactions. Apache Maven is a popular and broadly used tool for supporting the development of general software. In their contribution Adrian Paschke and Ralph Schäfermeier extend and customize the Maven approach for the distributed development and management of ontologies. A different approach for supporting software development is presented by Andrea Janes: data is collected during the software development process in a ubiquitous manner. The measurements are used to identify knowledge in the process data and to analyze the software creation and usage process.

In the past few years, a strong rise of interest in AI can clearly be observed in research, IT industry, as well as the general public. Today, the development of complex intelligent systems clearly benefits from the synergy of knowledge engineering and software engineering. Let this book, be a timely and valuable contribution to this goal.

Kraków, Poland Grzegorz J. Nalepa
Würzburg, Germany Joachim Baumeister
2017

Contents

Contributors

Klaus-Dieter Althoff Intelligent Information Systems Lab, University of Hildesheim, Hildesheim, Germany; Competence Center Case Based Reasoning, German Center for Artificial Intelligence, Kaiserslautern, Germany

Kerstin Bach Department of Computer and Information Science, Norwegian University of Science and Technology, Trondheim, Norway

Costin Badica University of Craiova, Craiova, Romania

Ralph Bergmann Business Information Systems II, University of Trier, Trier, Germany

Stamatia Bibi Department of Informatics & Telecommunications Engineering, University of Western Macedonia, Kozani, Greece

Marius Brezovan University of Craiova, Craiova, Romania

Paolo Ciancarini Department of Computer Science and Engineering, Consorzio Interuniversitario Nazionale per l'Informatica (CINI), University of Bologna, Bologna, Italy

Isabel M. del Águila University of Almería, Almería, Spain

José del Sagrado University of Almería, Almería, Spain

Adrian Giurca Brandenburg University of Technology Cottbus-Senftenberg, Cottbus, Germany

Wolfram Henkel Airbus Operations GmbH, Hamburg, Germany

Frieder Henning Lufthansa Industry Solutions, Hamburg, Germany

Andrea Janes Free University of Bozen-Bolzano, Bozen, Italy

Rainer Knauf Ilmenau University of Technology, Ilmenau, Germany

Angelo Messina Defense and Security Software Engineers Association, Innopolis University, Russian Federation, Rome, Italy

Gilbert Müller Business Information Systems II, University of Trier, Trier, Germany

José Neves Department of Informatics, Algoritmi Research Centre, University of Minho Braga, Braga, Portugal

Paulo Novais Department of Informatics, Algoritmi Research Centre, University of Minho Braga, Braga, Portugal

Tiago Oliveira National Institute of Informatics, Sokendai University, Tokyo, Japan

Adrian Paschke Computer Science Institute, Corporate Semantic Web Group, Freie Universität Berlin, Berlin, Germany

Pascal Reuss Intelligent Information Systems Lab, University of Hildesheim, Hildesheim, Germany; Competence Center Case Based Reasoning, German Center for Artificial Intelligence, Kaiserslautern, Germany

Francesco Poggi Department of Computer Science and Engineering, University of Bologna, Bologna, Italy

Daniel Russo Department of Computer Science and Engineering, Institute of Cognitive Sciences and Technologies, Italian National Research Council (CNR), Consorzio Interuniversitario Nazionale per l'Informatica (CINI), Rome, Italy

Ken Satoh National Institute of Informatics, Sokendai University, Tokyo, Japan

Ralph Schäfermeier Computer Science Institute, Corporate Semantic Web Group, Freie Universität Berlin, Berlin, Germany

Paraskevi Smiari Department of Informatics & Telecommunications Engineering, University of Western Macedonia, Kozani, Greece

Ioannis Stamelos Department of Computer Science, Aristotle University of Thessaloniki, Thessaloniki, Greece

Rotem Stram Competence Center Case Based Reasoning, German Center for Artificial Intelligence, Kaiserslautern, Germany

Part I
Methodological Studies

Aspect-Oriented Ontology Development

Ralph Schäfermeier and Adrian Paschke

Abstract Aspect-Oriented Ontology Development takes inspiration from Aspect-Oriented Programming and provides a novel approach to the problems of ontology modularization and metamodeling by adding support for reified axioms. The book chapter describes the syntax and semantics of Aspect-Oriented Ontology Development, explains its benefits and possible weaknesses as compared to other existing modularization approaches and presents a set of application scenarios as well as a set of supporting tools.

1 Introduction

This book chapter describes a novel approach to the problem of ontology modularization and re-use by the means of aspect-orientation [1].

With *Aspect-Oriented Ontology Development (AOOD)* we refer to a methodological approach, a set of logical formalisms and accompanying tools for modular ontology development and partial reuse of existing ontological knowledge based on requirements and cross-cutting concerns.[1] It is therefore a subfield of *Ontology Engineering* and a particular approach to the problem of *Ontology Modularization* and *Modular Ontology Development*.

Modular ontology development has proved beneficial when it comes to improvement of reasoning and query result retrieval performance, scalability for ontology

[1]As defined by the IEEE standard 1471 of software architecture [2], *"concerns are those interests which pertain to the systems development, its operation or any other aspects that are critical or otherwise important to one or more stakeholders"*.

R. Schäfermeier (✉) · A. Paschke
Freie Universität Berlin, Computer Science Institute, Corporate Semantic Web Group, Königin-Luise-Stra?e 24/26, 14195 Berlin, Germany
e-mail: ralph.schafermeier@gmail.com

A. Paschke
e-mail: paschke@inf.fu-berlin.de

G.J. Nalepa and J. Baumeister (eds.), *Synergies Between Knowledge Engineering and Software Engineering*, Advances in Intelligent Systems and Computing 626, https://doi.org/10.1007/978-3-319-64161-4_1

evolution and maintenance, complexity management, amelioration of understandability, reuse, context-awareness, and personalization [3]. A significant amount of research work has been dedicated in the field of ontology modularization, and various kinds of approaches tackle the problem from different perspectives. One kind of approaches provides algorithmic solutions for the problem of modularizing existing large and monolithic ontologies, while others provide methodological and formal means for contsructing ontologies in a modular fashion from scratch.

Aspect-Oriented Ontology Development is inspired by the *Aspect-Oriented Programming (AOP)*, also known as *Aspect-Oriented Software Development (AOOD)* paradigm.

2 Aspect-Oriented Programming

The main goal of Aspect-Oriented Programming is the decomposition of software systems into concerns which cross-cut the system. A code module covering a particular concern is referred to as an aspect. Concerns may be functional concerns, which are directly related to the system's domain of interest and business logic and non-functional concerns, such as security, logging/auditing and performance.

The decomposition is accomplished by introducing extensions to existing programming languages (such as AspectJ[2] for Java) that allow the decomposition of code into modules, each of them dealing with a concern, as well as a mechanism for recombining the modules at compile or runtime into a complete and coherent system.

Programming languages without aspect-orientation have no means for separating those concerns, which leads to undesired code tangling and hinders system decomposition.

2.1 Quantification and Obliviousness

Two principal properties of Aspect-Oriented Programming are *quantification* and *obliviousness* [4]. *Obliviousness* refers to the fact that all information necessary to determine the execution points where the application should make a call into an aspect module are contained within the aspect itself rather than in the application code. A developer of one module does not, and need not, have knowledge about other modules that might potentially be called.

This information may be provided in the form of an exhaustive list of signatures or in terms of *quantified statements* over signatures, called a *pointcut*. Each single matching signature is called a *join point*.

[2]https://eclipse.org/aspectj/.

Formally, Aspect-Oriented Programming uses quantified statements of the following form [5]:

$$\forall m(p_1, \ldots, p_n) \in M : s(sig(m(p_1, \ldots, p_n))) \\ \rightarrow (m(p_1, \ldots, p_n) \rightarrow a(p_1, \ldots, p_n)) \,, \qquad (1)$$

where M is the set of all methods defined in the software system, s a predicate specifying a matching criterion, $m(p_1, \ldots, p_n) \in M$ a method matching the signature $sig(m(p_1, \ldots, p_n))$, and $a(p_1, \ldots, p_n)$ the execution of the aspect with all the parameters of each method, respectively. The code in the aspect, which is executed at each joint point, is referred to as *advice*. In APO terminology, an aspect *advices* the main code.

The idea behind Aspect-Oriented Ontology Development is to use pointcuts in order to describe ontology modules and aspects in order to attach additional knowledge (advice) to each of these modules.

3 Aspect-Oriented Ontologies

The idea behind Aspect-Oriented Ontology Development is to use pointcuts in order to describe ontology modules and aspects in order to attach additional knowledge (advice) to each of these modules.

The semantics of ontology aspects are defined in correspondence with the possible-world semantics of multi-modal logics.

As in software, cross-cutting concerns can be observed in ontologies. Consider for example Abox facts that are constrained to be valid only during a certain period of time. Figure 1 shows a concrete example of a time-constrained fact, namely the recognition of the Kosovo as a self-governing entitiy, using concepts from the geopolitical ontology of the Food and Agriculture Organization of the United Nations.[3] The time period is modeled using the W3C time ontology.[4]

The intention is to reify the first fact recognizedBy(Kosovo, United_Kingdom) with the open time interval individual Interval_1 using a validDuring relationship. This, however, is not permissible due to limitations in the expressivity of OWL and the underlying Description Logics. What is instead recommended by the W3C is to introduce a surrogate individual to represent the ternary relationship between Kosovo, the UK and the time interval.[5] The right hand side of Fig. 1 shows the combined facts with the new introduced Recognition_1 surrogate individual. In addition to the individual, two new object properties need to be introduced. The existing recognizedBy property now has the new surrogate individual in its range

[3]http://www.fao.org/countryprofiles/geoinfo/en/.

[4]www.w3.org/TR/owl-time/.

[5]http://www.w3.org/TR/swbp-n-aryRelations/.

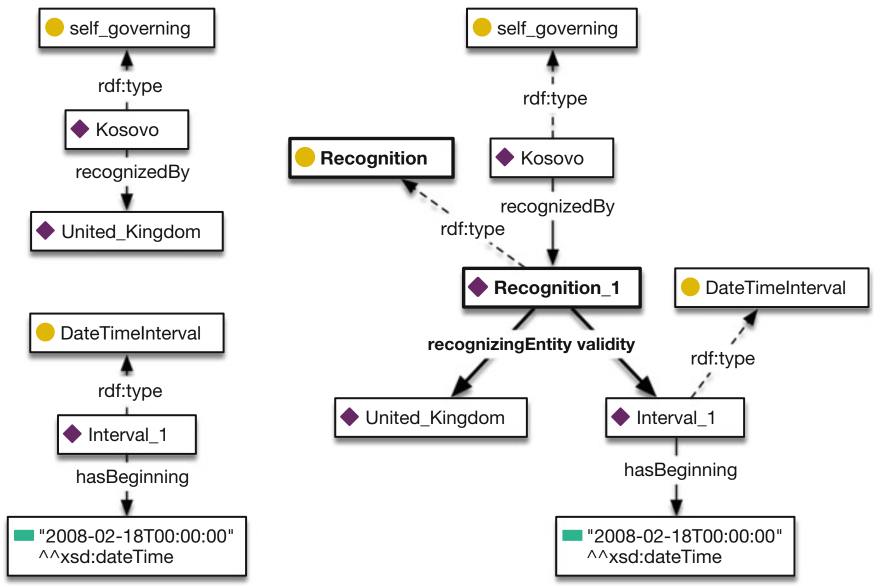


Fig. 1 Axioms from the FAO and the W3C time ontology (*left*) and necessary refactoring of the reused ontology in order to allow for the extension, following the W3C n-ary relations pattern (*right*)

instead of the recognizing country. The two new properties connect the surrogate with the recognizing country and the time interval, respectively.

The recommended pattern for n-ary relationships leads to a high degree of entanglement of different concerns (in this case different domains, namely the domain of self-governing political entities and the domain of time), which brings the following disadvantage: After the introduction of the surrogate individual, the representation of the fact recognizedBy(Kosovo, United_Kingdom) as a simple binary relation is lost. An ontology engineer, however, might be interested in reusing knowledge about self-governing entities from this ontology but without the temporal information. Due to the entanglement it is not trivial anymore to separate these parts from each other and reuse them individually.

Therefore, we introduce new syntactic category *Aspect*, which is used in order to establish a relationship between OWL 2 classes and axioms. Figure 2 depicts the representation of the ternary relationship from the above example using an aspect. Note that the figure contains two ternary relationships: The one from the example and the class assertion axiom Kosovo rdf:type self_governing which is also supposed to be valid only during the given time interval.

It might appear awkward to represent aspects as classes. In the following subsection where we describe the semantics of aspects, we provide a justification of that choice.

As can be seen, this way of representing the above relationships keeps the original structure of the ontology intact. In particular, it allows to connect additional

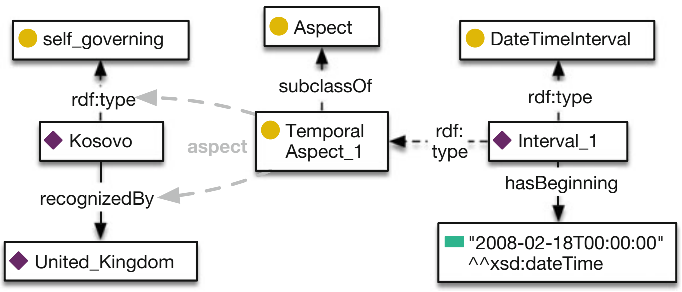


Fig. 2 Representation of the ternary relationships using an aspect. Note that the pointcut relation points to axioms, not individuals

(time) knowledge to an existing binary relation (about countries) while keeping both domains separated from each other, allowing partial reuse and independent maintenance and evolution.

3.1 Syntax

As mentioned above, we extend the OWL 2 language by a syntactic category *Aspect* which is used to represent a relationship between classes and axioms.

The definition of the abstract syntax in OWL functional-style syntax is as follows:

```
Aspect ::= 'Aspect' '(' Annotation∗ Advice ')'
AspectAssertion ::= 'AspectAssertion' '('
  AxiomAnnotationSet JoinPoint Advice ')'
AxiomAspectSet ::= Aspect∗
JoinPoint ::= IRI | AnonymousIndividual
Advice ::= ClassExpression
Pointcut ::=
  SPARQLPointcut | ModulePointcut | DLQueryPointcut
SPARQLPointcut ::= 'SPARQLPointcut' '('
  AxiomAnnotationSet Aspect '"' ConstructQuery '"' ')'
ModulePointcut ::= 'ModulePointcut' '('
  AxiomAnnotationSet Aspect Signature ')'
DLQueryPointcut ::= 'DLQueryPointcut' '('
  AxiomAnnotationSet Aspect ClassExpression ')'
Signature ::= EntityIRI∗
```

The definitions of the categories *Annotation, AxiomAnnotationSet, IRI, AnonymousIndividual, ClassExpression* and *Axiom* are provided in the OWL 2 Structural Specification.[6] The defintion of *ConstructQuery* is provided in the SPARQL 1.1 Query Language Specification.[7]

[6] http://www.w3.org/TR/owl2-syntax/.

[7] http://www.w3.org/TR/sparql11-query/#rConstructQuery.

An *AxiomAspectSet* can be added to each axiom that can contain an *AxiomAnnotationSet*, for example:

```
EquivalentClasses ::= 'EquivalentClasses' '('
 AxiomAnnotationSet AxiomAspectSet
 ClassExpressionSet ')'
```

3.2 Semantics

We define the semantics of ontology aspects in terms of two-dimensional interpretations, with one dimension accommodating the object logic and the other dimension accommodating the aspect meta-logic. An interpretation is a model for a sentence if the usual conditions on the object logic apply. A model on the aspect logic level is defined in correspondence to a Kripke model for modal logics, where a possible world represents a context in which a sentence of the object logic is true or not and classes of possible worlds can be used in order to define complex contexts using the available class constructors.

3.2.1 OWL Direct Semantics

The OWL 2 direct semantics as defined by the W3C[8] are compatible with the semantics of the Description Logic $\mathcal{SROIQ}$ plus datatypes, also referred to as $\mathcal{SROIQ}(\mathcal{D})$.

For the sake of simplicity, and without loss of generality, we neglect datatypes in our definitions here and only consider $\mathcal{SROIQ}$. The extension of $\mathcal{SROIQ}$ with datatypes is straighforward, and we refer the interested reader to [6].

Definition 1 Let $\mathsf{N_C}$, $\mathsf{N_R}$, and $\mathsf{N_I}$ be nonempty, pairwise disjoint sets of *concept names, role names,* and *individual names*. Then the triple $\mathsf{N} := (\mathsf{N_C}, \mathsf{N_R}, \mathsf{N_I})$ is called a *vocabulary*. The set of *concepts over* $\mathsf{N_C}$ is inductively defined using concept names $A \in \mathsf{N_C}$ and the constructors in the top section of Table 1, where $r, s \in \mathsf{N_R}$, $a, b \in \mathsf{N_I}$, $n \in \mathbb{N}$, and C, D are concepts over N. The syntax of axioms over N is shown in the bottom section of Table 1.

An *RBox* $\mathcal{R}$ *over* N is defined as a finite set of role inclusions, transitivity axioms, symmetry axioms, reflexivity axioms, role chain axioms, inverse role axioms, and disjoint role axioms over N.

A *Boolean axiom formula over* N is a combination of Boolean conjunctions and disjunctions of general concept inclusions, and concept and role assertions over N.

Finally, let $\mathcal{B}$ be a Boolean axiom formula over N and $\mathcal{R}$ an RBox over N. We then call the tuple $\mathfrak{B} = (\mathcal{B}, \mathcal{R})$ a *Boolean knowledge base (BKB) over* N.

Note that we adopt the above definition from [7], which derives from the classical separation of knowledge bases into an ABoxes, TBoxes and RBoxes. This separation

[8]https://www.w3.org/TR/owl2-direct-semantics/.

Table 1 Semantics of $\mathcal{SROIQ}$ constructors (top) and axioms (bottom)

	Syntax	Semantics
Top concept	$\top$	$\Delta_I^{\mathcal{I}}$
Negation	$\neg C$	$\Delta_I^{\mathcal{I}} \setminus C^{\mathcal{I}}$
Conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
Existential restriction	$\exists r.C$	$\{d \in \Delta_I^{\mathcal{I}} \mid$ there exists an $e \in C^{\mathcal{I}}$ with $(d, e) \in r^{\mathcal{I}}\}$
At-most restriction	$\leq_n r.C$	$\{d \in \Delta_I^{\mathcal{I}} \mid \#\{e \in C^{\mathcal{I}} \mid (d, e) \in r^{\mathcal{I}}\} \leq n\}$
nominal	$\{a\}$	$\{a^{\mathcal{I}}\}$
General concept inclusion	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
Concept assertion	$C(a)$	$a^{\mathcal{I}} \in C^{\mathcal{I}}$
Role assertion	$r(a, b)$	$(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$
Role inclusion	$r \sqsubseteq s$	$r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$
Transitivity	$\mathsf{trans}(r)$	$(d, e) \in r^{\mathcal{I}}$ and $(e, f) \in r^{\mathcal{I}} \rightarrow (d, f) \in r^{\mathcal{I}}$
Symmetry	$\mathsf{sym}(r)$	$(d, e) \in r^{\mathcal{I}} \rightarrow (e, d) \in r^{\mathcal{I}}$
Reflexivity	$\mathsf{ref}(r)$	$(d, d) \in r^{\mathcal{I}}$ for all $d \in \Delta_I^{\mathcal{I}}$
Role chain	$r \circ s$	$(d, e) \in r^{\mathcal{I}}$ and $(e, f) \in s^{\mathcal{I}} \rightarrow (d, f) \in (r \circ s)^{\mathcal{I}}$
Inverse role	$\mathsf{Inv}(r, s)$	$(d, e) \in r^{\mathcal{I}} \rightarrow (e, d) \in s^{\mathcal{I}}$
Disjoint roles	$r \sqcap s \sqsubseteq \neg\top$[9]	$r^{\mathcal{I}} \cap s^{\mathcal{I}} \subseteq \emptyset$

is not necessary in our work, and the generalization to Boolean knowledge bases results in more simplicity.

Also note that the top concept $\top$ may also be inductively defined as $A \sqcup \neg A$ with $A \in \mathsf{N_C}$ being arbitrary but fixed, where the disjunction $C \sqcup D$ is in turn defined as $\neg(\neg C \sqcap D)$. Furthermore, universal restrictions $\forall r.C$ may be defined as $\neg\exists r.\neg C$, at-least restrictions $\geq_n r.C$ as $\neg(\leq_{n-1} r.C)$ and the bottom concept $\bot$ as $\neg\top$. Finally, the concept equivalence axiom $C \equiv D$ may be expressed as the boolean conjunction of two general concept[9] inclusions $C \sqsubseteq D$ and $D \sqsubseteq C$.

Definition 2 Let $\mathsf{N} = (\mathsf{N_C}, \mathsf{N_R}, \mathsf{N_I})$ be a vocabulary, $\Delta^{\mathcal{I}}$ a nonempty set (called *domain*), and $\cdot^{\mathcal{I}}$ a mapping assigning a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ to every concept name $A \in \mathcal{N}_C$, a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ to every role name $r \in \mathsf{N_R}$, and an element of the domain $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ to every indiviual name $a \in \mathsf{N_C}$. The pair $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is an *interpretation over the vocabulary* N or simply an N-*interpretation*. Moreover, $\cdot^{\mathcal{I}}$ may be inductively applied to concepts over N as shown in the upper section of Table 1.

We call $\mathcal{I}$ a *model of an axiom* α *over* N if the respective condition in the lower section of Table 1 is satisfied. $\mathcal{I}$ is a model of a Boolean axiom formula $\mathcal{B}$ over N

[9]The bottom concept may be constructed by negation of the top concept, so we omit the dedicated bottom concept constructor in order to avoid redundancy. Likewise, we do not introduce the equvialent concept axiom, because it may be defined inductively from general concept inclusions and conjunction.

(denoted as $\mathcal{I} \models \mathcal{B}$) if for each of the axioms that it consists of the condition in the lower section of Table 1 is satisfied. Likewise, $\mathcal{I}$ is model of an RBox $\mathcal{R}$ over N (denoted as $\mathcal{I} \models \mathcal{R}$) if it is a model of each axiom in $\mathcal{R}$.

Finally, $\mathcal{I}$ is a model of the Boolean knowledge base $\mathfrak{B} = (\mathcal{B}, \mathcal{R})$ over N if it a model of $\mathcal{B}$ and if it is a model of $\mathcal{R}$.

3.2.2 Semantics of Modal Logics

Modal logics are propositional logics extended with two additional operators $\Box$ and $\Diamond$. Depending on the type of modal logic, the operators are named differently.

For example, in basic modal logic, $\Box$ is simply named *box*, and $\Diamond$ is named *diamond*. In logics that study necessity and possibility, $\Box$ is named *necessarily*, and $\Diamond$ is named *possibly*. In epistemic logics, $\Box$ stands for *"it is known that"*, and $\Diamond$ stands for *"it is believed that"*. In temporal logics, $\Box$ may stand for *"it has always been the case that"* or *"it will always be the case that"*, and $\Diamond$ may stand for *"it was the case that"* or *"it will be the case that"*, although it is common practice to use different symbols for the different types of logics.

Modal Logics are a syntactic variant of Description Logics [8]. This makes them easily combinable with DL based languages, since the languages' primitives may be mapped to either logic's features. Yet, the semantics of Modal Logics provide a more intuitive access to the notion of ontology aspects.

Definition 3 A *Kripke frame* is a tuple $\mathcal{F} = (W, R_i)$, where W is a set of *possible worlds*, and $R_i \subseteq W \times W$ an *accessibility relation* between worlds.

Furthermore, a *Kripke model* is a tuple (W, R_i, L), with W again a set of possible worlds, R_i an accessibility relation, and $L : W \to \mathcal{P}(Prop)$ a *labeling function*, where $\mathcal{P}$ is a *valuation function* $Prop \to \{T, F\}$ that maps propositional symbols to truth values. Moreover, let $\Box_i$ and $\Diamond_i$ be modal operators.

Then, the truth value of the propositional formula ϕ at a possible world w : $M, w \models \phi$ is defined in the following way:

- $M, w \models T$ and $M, w \not\models \bot$
- $M, w \models p$ iff $p \in L(x)$
- $M, w \models \neg\phi$ iff $M, w \not\models \phi$
- $M, w \models \phi_1 \vee \phi_2$ iff $M, w \models \phi_1$ or $M, w \models \phi_2$
- $M, w \models \phi_1 \wedge \phi_2$ iff $M, w \models \phi_1$ and $M, w \models \phi_2$
- $M, w \models \Box_i\phi$ iff $\forall w` \in W : wR_iw` \to M, w` \models \phi$
- $M, w \models \Diamond_i\phi$ iff $\exists w` \in W : wR_iw` \to M, w` \models \phi$

Furthermore, by adding certain axioms it is possible to alter the behavior of the logic and obtain a particular type of modal logic. According to the *correspondence theorem*, each axiom corresponds to a particular condition which is imposed on the frame. This in turn corresponds to the characteristic of the accessibility relation of the frame. Table 2 shows the correspondences between logic types, axioms, conditions on frames, and characteristics of the accessibility relation.

Table 2 Correspondence between types of modal logic, axioms, conditions on frames, and characteristics of the accessibility relation

Name	Axiom	Condition on frames	R is…
(D)	$\Box\phi \rightarrow \Diamond\phi$	$\exists u wRu$	Serial
(M)	$\Box\phi \rightarrow \phi$	wRw	Reflexive
(4)	$\Box\phi \rightarrow \Box\Box\phi$	$(wRv \wedge vRu) \rightarrow wRu$	Transitive
(B)	$\phi \rightarrow \Box\Diamond\phi$	$wRv \rightarrow vRw$	Symmetric
(5)	$\Diamond\phi \rightarrow \Box\Diamond\phi$	$(wRv \wedge wRu) \rightarrow vRu$	Euclidean
(CD)	$\Diamond\phi \rightarrow \Box\phi$	$(wRv \wedge wRu) \rightarrow v = u$	Functional
$(\Box M)$	$\Box(\Box\phi \rightarrow \phi)$	$wRv \rightarrow vRv$	Shift Reflexive
$(C4)$	$\Box\Box\phi \rightarrow \Box\phi$	$wRv \rightarrow \exists u(wRu \wedge uRv)$	Dense
(C)	$\Diamond\Box\phi \rightarrow \Box\Diamond\phi$	$wRv \wedge wRx \rightarrow \exists u(vRu \wedge xRu)$	Convergent

Table 3 Correspondence between modal logics and Description Logics

Modal logics	Description logics
$M, w \models \neg A$ iff $M, w \not\models A$	$\neg C^{\mathcal{I}} = D^{\mathcal{I}} \setminus C^{\mathcal{I}}$
$M, w \models A_1 \wedge A_2$ iff $M, w \models A_1$ and $M, w \models A_2$	$(C_1 \sqcap C_2)^{\mathcal{I}} = C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
$M, w \models A_1 \vee A_2$ iff $M, w \models A_1$ or $M, w \models A_2$	$(C_1 \sqcup C_2)^{\mathcal{I}} = C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$
$M, w \models \Box_i A$ iff $\forall w` \in W : wR_i w` \rightarrow M, w` \models A$	$(\forall r.C)^{\mathcal{I}} = \{x \mid \forall y((x, y) \in r^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}})\}$
$M, w \models \diamond_i A$ iff $\exists w` \in W : wR_i w` \rightarrow M, w` \models A$	$(\exists r.C)^{\mathcal{I}} = \{x \mid \exists y((x, y) \in r^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}})\}$

3.2.3 Correspondence Between Modal and Description Logics

As mentioned in Sect. 3.2.2, there exists a direct correspondence between Description Logics and modal logics (see table 3).

3.2.4 Semantics of Ontology Aspects Using Two-Dimensional Interpretations

We can now define the semantics of ontology aspects. The intuition of an aspect as laid out in Sect. 3 is to provide external context knowledge in order to specify circumstances in which a part of the knowledge encoded in an ontology holds and in which it does not.

We define a combined interpretation, which we call a $\mathcal{SROIQ}_{Kripke}$ interpretation. The idea is based on nested interpretations as introduced by Böhme and Lippmann [9] with the difference that we combine two different kinds of models.

Definition 4 A $\mathcal{SROIQ}_{Kripke}$ interpretation is a tuple $\mathcal{J} := (W, R, L, \cdot^{\mathcal{J}}, \Delta, (\cdot^{\mathcal{I}_w})_{w \in W})$ with W being a nonempty set, called *possible worlds*, and L a Kripke interpretation, assigning truth values to propositional symbols in each world $w \in W$ as described in Sect. 3.2.2.

For every $A \subseteq W$, $\mathcal{I}_A$ is a DL interpretation.

The semantics of an aspect of an axiom is then defined as follows:

Definition 5 Let $\mathcal{J} := (W, R, L, \cdot^{\mathcal{J}}, \Delta, (\cdot^{\mathcal{I}_w})_{w \in W})$ be a possible-world DL interpretation. We interpret an aspect under which an axiom α holds as follows:

$(\mathsf{hasAspect}(\alpha, A))^{\mathcal{J}} \rightarrow A^{\mathcal{J}} \subseteq C^{\mathcal{J}} := \{w \in W \mid \mathcal{I}_w \models \alpha\}$. Because of the correspondence as described in Sect. 3.2.3 we can set $W = C^{\mathcal{J}}$, such that on the semantic level each individual corresponds to a possible world. Furthermore, we set L such that $L(\alpha)^{\mathcal{J}} := A^{\mathcal{J}}$.

The intuition is to have an aspect correspond to a propositional formula that might or might not be mapped to $True$ by L in a particular possible world w.

Due to the correspondence between Description Logics and modal logics, a proposiotional formula may as well be interpreted as a concept C and the worlds in which the formula is mapped to $True$ by L as indivudials which are instances of C. The accessibility relations may be interpreted as roles. The choice of characteristics of the roles determines the type of modal logic and thereby the semantics of the modal operators.

This yields the following advantages:

- Aspects correspond to sets of axioms or facts that are true in certain possible worlds.
- Aspects are modeled as classes.
- Possible worlds are modeled as individuals.
- Accessibility relations are modeled as object properties.
- The semantics of aspects depend on the choice of conditions on frames (axioms on accessibility properties).

The rationales behind that choice are:

- (Multi-)modal logics are a syntactic variant of and thereby semantically equivalent to Description Logics [7, 8].
- Aspects are a sort of modality in that there is a function that determines in which situations an aspect is active and in which it is not. That corresponds to possible worlds in modal logics where a truth-functional valuation determines whether a fact is valid in a possible world or not.
- The kind of modal logic is determined by conditions on Kripke frames, which (to a certain extent) may be controlled by fixing the characteristics of the accessibility relations. This allows the representation of e.g., temporal logic (as in our running example), simple views, agent beliefs, etc.
- Using classes as aspects allows to use abstract class definitions using constraints with quantifiers.

Figure 3 depicts a more complex example using a temporal aspect on an Abox fact $\mathsf{capitalOf(Bonn, Germany)}$, which was true between 1949 and 1990. We used the W3C time ontology again to model time instances. We interpret each time instance as a possible world, and *after* (and *before*) are accessibility relations, which are reflexive and transitive. We thereby obtain the conditions on the Kripke frames for a temporal logic:

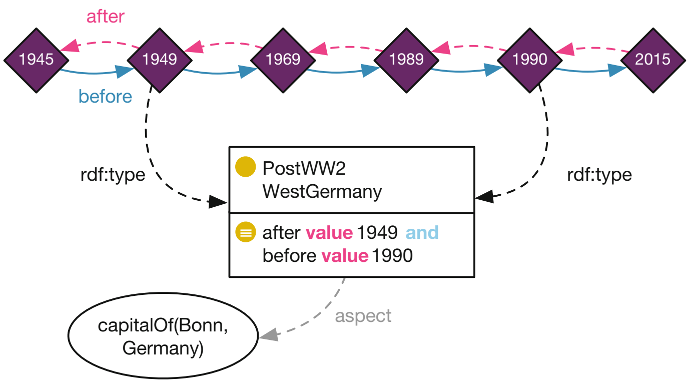


Fig. 3 A temporal aspect using temporal logic

- $(M) : \Box A \rightarrow A$
- $(4) : \Box A \rightarrow \Box\Box A$

The temporal aspect is then the class expression
after value 1949 and before value 1990,
which includes the values 1949 and 1990 due to the reflexivity of the before and after relations.

Likewise, we can obtain a simple Logic K by just setting the accessibility relation reflexive. We can use this logic to model simple views, which are manually assigned to axioms.

As a third example, we can use multi standard deontic logic for modelling access permissions over axioms for different agents by having a serial accessibility relation a_i for each agent i in order to obtain the axiom

- $(D) : \Box_i A \rightarrow \Diamond_i A$

The intuition behind this is that an aspect describes a (syntactic) module in an ontology (which technically consists of a set of axioms) and adds second-order information to it (as for example a temporal validity restriction, as in the above example). The purpose of this approach is to permit to extract modules depending on the outcome of some reasoning process. We could, for example, extract a module with axioms that are valid only during the 1950s (which would include the fact that Bonn is capital of Germany) and at the same time are accessible to some agent.

Defining ontology aspects as meta-statements which describe possible worlds in which an ontology axiom may be either true or false allows for a number of industrial as well as research scenarios.

4 Application Scenarios

In the context of the Corporate Smart Content research project we implemented the following research and industrial scenarios using aspect-oriented ontologies.

4.1 Research Scenarios

- Simple views
  Aspects in their simplest form provide simple context information with attach to ontology axioms or facts.
  Simple views are realized using a simple Logic K with only one condition on the modal frames. With Logic K it is possible to express:
  - named views
  - provenance information
  - meta-data of any type
  - references to other named resources, e.g. for
    - multi-faceted alignment to external taxonomies
    - optional, non-normative descriptions
- Inconsistent knowledge
  Logic K is an epistemic logic and relies on the fact that knowledge is consistent across the boundaries of possible worlds. This scenario also describes cases in which the axiomatization of a domain may change depending on an externally provided context, but, unlike in the first scenario, inconsistencies in the knowledge in different contexts is allowed.
  An example of an inconsistency would be two contradicting axioms in the same ontology, each of which represents the view of a particular stakeholder.
  A possible way of reasoning with inconsistent knowledge is resorting to paraconsistent logic and consists in preventing the explosion at the cost of abandoning one of the three principles disjunction induction ($A \vdash A \vee B$), the disjunctive syllogism ($A \vee B, \neg A \vdash B$) or transitivity ($\Gamma \vdash A; A \vdash B \Rightarrow \Gamma \vdash B$).
  With aspect-oriented ontologies this reduction is not necessary, since the mutually contradicting axioms are contextualized. It is only necessary to keep the possible worlds representing the contexts isolated from each other by forbidding accessibility relations between them.
- Multi-agent belief using doxastic logic
  A further conceivable scenario involving the deployment of semantics similar to that of logic K is the interpretation of contexts as agents' beliefs about axioms and facts in a knowledge base. Doxastic Logic uses a modal operator $\mathcal{B}$, which expresses the belief of an agent in axioms and facts. By indexing the belief operators, we obtain a multi-modal logic with multiple belief operators, each representing the belief of one particular agent.
- Temporal aspects using temporal logic
  While ontologies are supposed to capture universally valid knowledge, there are situations in which an application might make use of knowledge that is contextualized with time. A problem that arises in making universal statements in the form of *A is valid during time interval T* is that actual domain knowledge and contextual (time) knowledge become tangled.

Using temporal aspects, each time instance is interpreted as a possible world, and *after* (and *before*) are accessibility relations, which are reflexive and transitive. The corresponding conditions on the Kripke frames for a temporal logic are:

- $(M) : \Box A \rightarrow A$
- $(4) : \Box A \rightarrow \Box\Box A$

The temporal aspect is then the class expression
after value 1949 and before value 1990,
which includes the values 1949 and 1990 due to the reflexivity of the before and after relations.

- Access Rights using Deontic Logic
  Controlling access to sensitive knowledge contained in knowledge bases is important especially in corporate contexts. Currently, access restrictions are part of the technological infrastructure, for example, SPARQL endpoints may be accessible only after authentication.
  With aspects it is possible to model access restrictions in a more fine-grained manner, by providing arbitrarily many access permissions for each axiom or fact in the knowledge base.
  The problem with restricting access to parts of knowledge in a global model is that agent who are forbidden to access certain parts of the model have to deal with an incomplete model of the world. This may (not necessarily but possibly) lead to inconsistent states in the local world view of such an agent.
  By modeling access restrictions as aspects using deontic logic, it is possible to model an alternative sub model for an access restricted part of the global model. It is possible to model the "forbidden" state not as failure but rather provide an alternative, simplified truth. This may be done by providing a set of "repair axioms" that complete the world view of the agent in such a way that it becomes consistent again.
  Multi standard deontic logic for modelling access permissions over axioms for different agents by having a serial accessibility relation a_i for each agent i in order to obtain the axiom

  - $(D) : \Box_i A \rightarrow \Diamond_i A$

4.2 Industrial Scenarios

- Development of Multilingual Ontologies
  A recurring challenge in cross-cultural contexts is multilingualism. Examples of situations requiring multilingual assessment of knowledge include global corporations with subsidiaries in different countries that desire to build a company-wide body of knowledge, or business intelligence scenarios addressing international markets. Another example are multilingual web applications with databases of

goods that, depending on the country have different names but also different components or ingredients, like for example a drug database with the same medication having different ingredients due to different law or patent situations in the different countries.
The challenge in this scenario goes beyond the need to attach multilingual labels to concepts and objects. Instead, the intensional description of entire concepts or objects may differ completely.
This scenario requires a contextualized conceptualization of the domain of interest, where language or cultural background constitutes the context. It possibly needs to deal with global inconsistencies due to locally consistent but globally contradicting world views.

- Role-Specific Views on Business Processes
  This scenario is based on the conceptualization of entities relevant in a cross-department business process model in a production environment. The process model vocabulary includes

  - 195 concepts and
  - 52 properties

  These may or may not be relevant in 12 contexts which represent the view of different stakeholders as well as 3 meta contexts representing technical aspects of the process context.
  The contexts are *Context, Constructor, Service, Controller, Procedure, Transport, Component, Procedure Map, Request Checklist, Facility, Offer, Client, Documentation, Inquiry, Problem,* and *Glossary*.
  These contexts can be modeled independently of the domain concepts and properties and then later combined.
- Street names in location-based search
  This application scenario involves a search facility for historical information about locations in Berlin including semantic descriptions of the localities, which are, among others, provided by modeling streets and their names.
  One problem that needs to be addressed is that street names in Berlin have undergone a significant amount of change during the last century due to natural growth of the city, but also due to its dynamic history.
  Since the goal of this scenario is change of concepts over time, it is a candidate for the application of temporal aspects. As in the general scenario description for temporal aspects, this scenario does not only include the changing of labels but also attributes of entities (many streets kept their name after the wall was built, but were cut in half by the wall, so that only a part of the original street kept the original name, while the other half became a new street with a new name). Also the conceptualization changes slightly since categories representing historical periods (e.g. "Street in East Berlin") which in themselves only exist during certain periods of time are deployed.

- Access Rights in Corporate Wiki using Deontic Logic
  Internal content may be contextualized by access restrictions to certain groups of coworkers.
  This scenario covers the above problem in the context of internal corporate wiki systems combined with a knowledge extraction task. It integrates into the authentication and access-rights management system of the wiki.
  Users create content in the wiki. While doing so, they fulfill one or more roles in the enterprise. The content they provide is access-restricted based on the enterprise specific role model.
  A knowledge extraction process creates axioms and facts representing the knowledge stored in the wiki.
  This scenario requires a model which aids in applying the same access restrictions that apply to the textual content to the factual knowledge gained by the extraction process. It is a candidate for the application of the general access rights research scenario.
- Temporal Attribution of Facts in a Corporate Wiki
  This scenario addresses a different problem in the same wiki application described above.
  Corporate knowledge is dynamic. As users add content to the wiki, knowledge my become stale (i.e., still valid but not useful anymore) or invalid and replaced by new, possibly contradicting knowledge. This is a use case for using temporal aspects as described in the research scenarios.

5 Tools

Aspect-Oriented Ontology Development is facilitated by a set of tools which will also be presented in the following. They comprise:

- Aspect-oriented OWL API: an extension to the OWL API for transparent programmatic access to aspect modules in ontologies using java annotations,
- Aspect OntoMaven: Integration of aspect-based module extraction into the ontology lifecycle management tool OntoMaven,
- Editors: Extensions to the ontology editors Protg and WebProtg provide ontology developers support in creating and using aspects, defining query based pointcuts, using aspects as editing contexts, as well as extraction of modules based on aspects and reasoning with aspects.

5.1 An Annotation-Based API for Programmatic Access to Aspect-Oriented Ontologies

Applications using ontologies come with a set of requirements. A subset of these requirements are related to how the ontology is going to be used and what the application developers expect to get back from the ontology. These requirements are in fact related to the ontology itself, still they concern the application's mode of operation. As those particular requirements affect both the application and the ontologies in the same manner, it appears natural to have them reflected in a unified way in the application code as opposed to have them scattered across the application. For this reason, we have developed a software artifact that makes ontology aspects accessible to developers of semantic web applications [10]. We have implemented it in the form of an aspect-oriented extension to the well-known OWL API,[10] a Java API for OWL 2 ontologies, using AspectJ[11] as a Java aspect language and Java annotations as a way to declaratively control ontology aspects from Java code.

The most notable APIs for web ontologies are Apache Jena[12] and the OWL API.[13] With both being Java APIs, Jena's scope are RDF graphs while the OWL API is tailored to the OWL language,[14] providing an object model for OWL entities and axioms and interfaces for DL-based reasoning and explanation support. In what follows, we describe an extension of the OWL API by programmatic access to OWL aspects as defined in the previous sections.

5.1.1 Requirements

We defined the functionality of our approach in terms of functional requirements, which can be seen in Table 4.

We designed an aspect-oriented extension of the OWL API using the Java aspect language AspectJ.[15] We use Java aspects in order to intercept read/write access to the OWL API's Java object model of a loaded OWL ontology and advice the calls by code responsible for extracting ontology modules affected by an OWL aspect.

With regard to requirement 2 we decided to use Java annotations as a means to convey the selected aspects from the Java side.

[10]http://owlapi.sourceforge.net/.
[11]https://eclipse.org/aspectj/.
[12]http://jena.apache.org/.
[13]http://owlapi.sourceforge.net/.
[14]At the time of the writing of this report, the target language of the OWL API was OWL 2.
[15]https://eclipse.org/aspectj/.

Table 4 Functional requirements for the OWL API extension with ontology aspects

ID	Name	Description
F-1	Aspects	The system should allow for mapping aspects to ontology modules.
F-1.1	Aspect declaration	The system should permit the mapping of declarative aspect descriptions to OWL aspects in the set of loaded ontologies.
F-1.1.1	Aspect identification	Aspects should be ontological entities identified by IRIs, as defined in the AspectOWL ontology.
F-1.1.2	Aspect combination	The declarative selection of aspects should allow for a combination of multiple aspects using logical AND/OR operations.
F-2	Ontology change	If an aspect declaration is associated with code manipulating an ontology, then the manipulation must only affect this aspect
F-2.1	Axiom addition	If an axiom is added, then it will be annotated with the aspect(s) present in the declaration.
F-2.2	Axiom deletion	If an axiom is deleted, and an aspect declaration is present in the Java code, then only the corresponding aspect associations must be deleted in the ontology (the axiom is only removed from this aspect, not from the ontology).
F-3	Module extraction	If an aspect declaration is associated with code reading from an ontology, then only that part of the ontology is returned, which is associated with the given aspects. The rest of the ontology is hidden.

```
@OWLAspect({"http://www.fu-berlin.de/csw/ontologies/aood/
    ontologies/aspect123", "http://www.fu-berlin.de/csw/
    ontologies/aood/ontologies/myAspect456"})
public void doSomething () {
 Set<OWLAxiom> allAxioms = myOntology.getAxioms();
 ...}
```

Listing 1 Example client code using the OWL API. Note that aspect references are added transparently and uninvasively as method annotations. Client code itself does not need to be changed.

The heart of the extension is a set of pointcut definitions that intercept all calls to the OWL API that either return or manipulate OWL entities or axioms. Client code using the OWL API can use the `@OWLAspect` java annotation type to specify one ore more aspect IRIs either on the method or class level. If one or more annotations of that type are encountered, then all operations on the OWL API within the context of the annotation will only be executed on the subset of the ontology that corresponds to the aspect(s) specified by the annotations. For example, a call to `OWLOntology.getAxioms()` from a client method with the annotations from Listing 1 would only return those axioms that belong to the modules specified by the two IRIs used in this example. Accordingly, write operations would also only be performed on the subset.

5.1.2 Conjunctive and Disjunctive Combination of Aspects

In order to fulfill requirement F-1.1.2, the annotation type system had to be extended by some kind of boolean arithmetics that allows for conjunctive and/or disjunctive combination of multiple aspects. The Java annotations system does not permit relations between single annotations in the same place, but nesting of annotations is possible. In order to provide a syntax for boolean operations, we subclassed the OWLAspect annotation type by OWLAspectAnd and OWLAspectOr.

Unfortunatley, nesting of annotations is restricted to non-cyclic nesting levels, disallowing arbitrary nesting (and thereby arbitrary boolean combinations). However, using the distributivity property of boolean operations, it is possible to bring every boolean formula into a non-nested form. As a consequence, this approach is feasible for arbitrary boolean combinations of aspect declarations conveyed using Java annotations.

An example of a combination would be:

```
@OWLAspectOr({
 @OWLAspectAnd({"http://...#Aspect1",
  "http://...#Aspect2"}),
 @OWLAspectAnd({"http://...#Aspect2",
  "http://...#Aspect3"})
})
```

5.1.3 Syntactic Verses Semantic Modules

The above approach allows for the selection of subsets of axioms, also referrred to as syntactic modules as mentioned in the introduction. Depending on the use case, this might be sufficient, or it might be necessary to extend the selected set of axioms to a proper semantic module, such that the parent ontology is a conservative extension of the module. The system we present here may extend the selected subset to a proper module by using syntactic locality [11] as an approximation.

5.2 Aspect OntoMaven

OntoMaven[16] is a Maven-based tool for ontology management based on distributed ontology repositories [12] (see also Chap. 4). The lifecycle of an ontology in a project and its dependencies can be managed in a declarative way via Maven's Project Object Model (POM). OntoMaven's ontology specific functionality is provided in the form

[16]http://www.corporate-semantic-web.de/ontomaven.html.

of plug-ins to the Maven framework. It includes dependency resolution, ontology unit testing, documentation and visualization.

The extension of OntoMaven with aspect-oriented concepts allows the declarative configuration and automated interweaving of ontology aspects during the build phase.

5.2.1 OntoMvnApplyAspects

This plug-in contributes to the `package` goal of Maven's build lifecycle. It takes the ontologies that are part of the OntoMaven project and selects exactly those modules that are specified by the Maven parameter `userAspects`. An additional parameter `aspectsIRI` allows to specify a custom OWL object or annotation property which is used to map the aspects to the ontology modules. The parameter `ifIncludeOriginalAxioms` specifies whether those axioms in the ontology that are free of aspects (the *base module*) should be included in the resulting ontology or not. This allows to either configure an ontology and enable selected aspects of it or to merely extract a module on its own and use it in the application. An example configuration is shown in the following POM listing:

```
<build> <plugins> <plugin>
    <groupId>de.csw.ontomaven</groupId>
    <artifactId>OntoMvnApplyAspects</artifactId>
    <version>1.0-SNAPSHOT</version>
    <configuration>
        <userAspects>
            <aspect>http://example.org/reputation#Reputation123</aspect>
            <aspect>http://example.org/provenance#prov_789</aspect>
        </userAspects>
        <aspectsIRI>http://corporate-semantic-web.de/aspectOWL#hasAspect
        </aspectsIRI>
        <includeOriginalAxioms>true</includeOriginalAxioms>
    </configuration>
...
```

5.3 Editors and Aspect Management Tools

In order to facilitate the the development and integration of aspect-oriented ontologies, two editing and management tools have been implemented.

The first one consists of an extension to the well-known Protégé editor.

5.3.1 Desktop Protégé

As described in Sect. 3, the connections between aspects and their targets may either established by using an extensional description (by providing a complete set of targets), or intensionally (by specifying the properties of the targets using an abstract

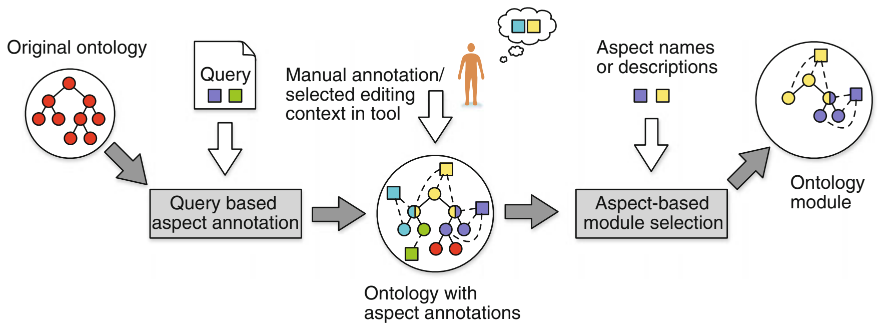

Original ontology
Query
Manual annotation/ selected editing context in tool
Aspect names or descriptions
Query based aspect annotation
Ontology with aspect annotations
Aspect-based module selection
Ontology module

Fig. 4 Our approach to aspect-oriented ontology modularization. Axioms of the original ontology (*left*) are annotated with entities from an external aspect ontology (*center*). Axiom selection is based on queries or is performed manually. Module extraction happens dynamically, as a particular part of the ontology is requested (*right*). The parameters of the request are one or several aspect names. The result is an ontology module including exactly those axioms that belong to the given aspects

query or quantification). Functionality for the first variant involves manually creating the appropriate annotation axioms (see Fig. 4, middle).

The intensional specification by quantification is realized through a SPARQL interface. Using SPARQL CONSTRUCT queries, the subgraph of the triple based ontology representation that corresponds to the desired axioms is selected. The resulting subgraph is transformed back into an OWL 2 model, which then contains the desired set of axioms. For each axiom in the result set, an aspect annotation axiom is then added to the ontology, with the axiom as annotation subject and the specified aspect as the annotation object (see Fig. 4).

Module Extraction

The extraction process involves providing a set of aspects and results in the extraction of the corresponding ontology modules (see Fig. 4, to the right). The provided aspects can be the result of manual selection or of a query, e.g., by using the DL-query facility provided by Protégé.

Aspects and Entailment

The checking for aspects of entailed axioms has been implemented in the following way:

If an ontology contains asserted axioms that are intended to hold only under particular aspects, then the question arises under which aspects the logical consequences of these axioms hold. In order to determine that, we can employ justifications for entailments as defined by Horridge et al. in [13].

Definition 6 *(Justification)* For an ontology $\mathcal{O}$ and an entailment η where $\mathcal{O} \models \eta$, a set of axioms $\mathcal{J}$ is a justification for η in $\mathcal{O}$ if $\mathcal{J} \subseteq \mathcal{O}$, $\mathcal{J} \models \eta$ and if $\mathcal{J}' \subsetneq \mathcal{J}$ then $\mathcal{J}' \not\models \eta$.

As also noted in [13], there may be multiple, potentially overlapping, justifications for a given entailment. It is obvious that if there exists an explanation for an entailment that only contains axioms without any associated aspects, then the entailment is also free of aspects.

In cases where multiple explanations for an entailment exist, one of which contains an axiom with an aspect and another one contains an axiom with a different aspect, then the entailment has both aspects. Algorithm 1 demonstrates the carrying over of aspects to inferred axioms.

Algorithm 1 Carrying over aspects from asserted to inferred axioms using justifications.

```
Input: An ontology $\mathcal{O}$, an aspect ontology $\mathcal{O}_A$ an entailment $\eta$ with $\mathcal{O} \models \eta$ and the set $\{\mathcal{J}_1, \ldots, \mathcal{J}_n\}$
    of all justifications for $\eta$.
 1: A $\leftarrow \emptyset$
 2: for all $\mathcal{J} \in \{\mathcal{J}_1, \ldots, \mathcal{J}_n\}$ do
 3:     flag $\leftarrow$ $false$
 4:     for all ex $\in \mathcal{J}$ do
 5:         for all a $\in \mathcal{O}_A$, asp(ex a), a $\preceq$ hasAspect do
 6:             A.$add$(asp($\eta$ a))
 7:             flag $\leftarrow$ $true$
 8:         end for
 9:         if flag == $false$ then return
10:         end if
11:     end for
12: end for
13: $\mathcal{O} \leftarrow \mathcal{O} \cup$ A
```

The set A and the boolean flag are needed for determining whether there exists a justification $\mathcal{J}$ containing only axioms that are free of aspects. For each $\mathcal{J} \in \{\mathcal{J}_1, \ldots, \mathcal{J}_n\}$, where any of the axioms ex is assigned to an aspect a using the property asp, which is a sub-property of a property hasAspect, the aspect assigning axiom is added to a set A replacing ex with η (line 6) and the flag is set to $true$ (line 7). Once the inner loop terminates, and the flag still has the value $false$, then the inferred axiom can safely be considered aspect free, and the algorithm can terminate prematurely (line 9). Otherwise, all aspect assigning axioms are collected in A and eventually added to $\mathcal{O}$ (line 13).

It must be noted that if aspects are used for modularization of an ontology, and it is explicitly allowed to have contradictions in the original ontology (which are meant to be resolved during the module selection stage), if the above algorithm is applied to the original ontology, it needs to be assured that conflicting information is not carried over to the selected modules later.

Given, e.g., an ontology consisting of the following axioms:

capitalOf(Rio_De_Janeiro, Brazil), capitalOf(Brasília, Brazil),
hasTemporalAspect((capitalOf(Rio_De_Janeiro, Brazil)), timeInterval1),
hasTemporalAspect((capitalOf(Brasília, Brazil)), timeInterval2),

```
  startDate(temporalAspect1,    "1889-11-15T00:00:00"^^xsd:date
Time"),
  endDate(temporalAspect1,      "1960-04-21T00:00:00"^^xsd:date
Time"),
  startDate(temporalAspect2,    "1960-04-21T00:00:00"^^xsd:date
Time"),
  InverseFunctionalObjectProperty(capitalOf).
```

The ontology represents the fact that the republican Brazil had different capitals throughout its history, namely Rio de Janeiro from 1889 to 1960, and Brasília from 1960 onwards. The time intervals are represented using aspects, with the consequence that the original ontology contains both the axioms capitalOf(Bonn, Germany) and capitalOf(Berlin, Germany).

The additional InverseFunctionalObjectProperty(capitalOf) axiom will lead to the unwanted entailment sameAs(Rio_De_Janeiro, Brasília). In such cases, it is necessary to make the intended semantics explicit, e.g., as in the above example, by adding an axiom differentFrom(Rio_De_Janeiro, Brasília).

5.4 An Aspect Weaver for Ontologies Using Structural Ontology Design Patterns

In this section, we introduce a weaving facility for ontologies, which converts aspect-oriented ontologies into standard-conformant OWL 2 ontologies, preserving the information conveyed by the aspects.

In Aspect-Oriented Programming, which also requires an extension of the programming language at hand for the representation of software aspects, a special software utility named *aspect weaver* is responsible for recombining the modules using the extra information contained in the aspect extension and generating executable code that conforms to the standards of the programming language. We use ontology design patterns (ODPs) in order to automate the conversion process. ODPs are structural (e.g., logical or architectural), conceptual, or lexico-syntactic templates that abstract from typical ontology modeling problems and serve as a recipe for ontology developers to solve the corresponding problem.

The weaver presented here uses the three patterns *View Inheritance, Context Slices,* and *N-ary Relation Pattern*. We used an extended version of the Ontology Pre-Processor Language (OPPL)[17] for formulating the patterns and the necessary refactoring operations. We extended OPPL by syntactical features for aspects, in correspondence to the aspect-oriented extensions of OWL 2 described in Sect. 3.1.

The *View Inheritance ODP*[18] is an architectural pattern that provides a way for modeling multifacted classification schemes or multiple class inheritance hierarchies.

[17]http://oppl2.sourceforge.net.

[18]http://ontologydesignpatterns.org/wiki/Submissions:View_Inheritance.

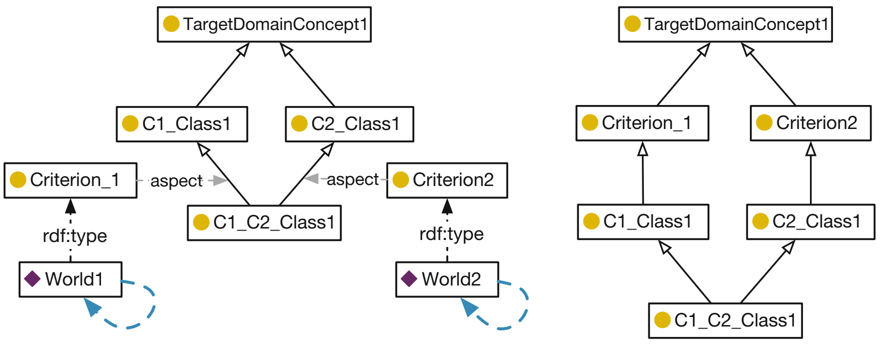


Fig. 5 A transformation by the weaver of multifaceted classification using the View Inheritance ontology design pattern. The aspect is expressed by a reflexive accessibility relation in a logic system K. The class expression representing the aspect is attached to the subclass axioms

It does this by introducing intermediate classes which represent the different classifiers, referred to as *criteria*. The actual target domain concepts are made subclasses of the classes representing the classification criteria. The pattern has two disadvantages, a semantic and a structural one. The semantic disadvantage consists in the fact that the classifier classes are directly introduced into the inheritance hierarchy. Subclasses are now subclasses of the classifier, which is not the intended meaning but merely a way to circumvent the expressive restrictions of DL which do not allow object relations between classes. The structural disadvantage is that once introduced, the classifier cannot easily be eliminated from the hierarchy. A typical use case for multifaceted classification is to specify a classifier and hide the other hierarchies with different classifiers. Since the classifier is now part of the ontology, this is not easily possible. In an aspect-oriented ontology, this kind of classifier is represented as an aspect class which is attached to the owl:SubClassOf axioms that correspond to this specific classifier. Since it expresses simple views, Logic K is the appropriate type of modal logic, and therefore, this aspect has a reflexive accessibility relation.

Figure 5 shows how the weaver transforms the aspects into an ontology by applying the pattern.

The *context slices* pattern[19] may be used for the expression of agents' beliefs about Abox facts, or, more precisely, object property assertions. Each agent's conception of a part of the universe (i.e., the axioms valid in the part of the universe accessible to the agent) is referred to as a context. A context is represented by an individual of type Context, and the subject and object of a contextualized object property assertion are connected to this context via additional object property assertions involving an object property hasContext with the context individual in the subject role and the two contextualized individuals in the object role.

In an aspect-oriented ontology the belief of an agent may be expressed using a doxastic logic with an object property believes_i, representing the accessibility

[19]http://ontologydesignpatterns.org/wiki/Submissions:Context_Slices.

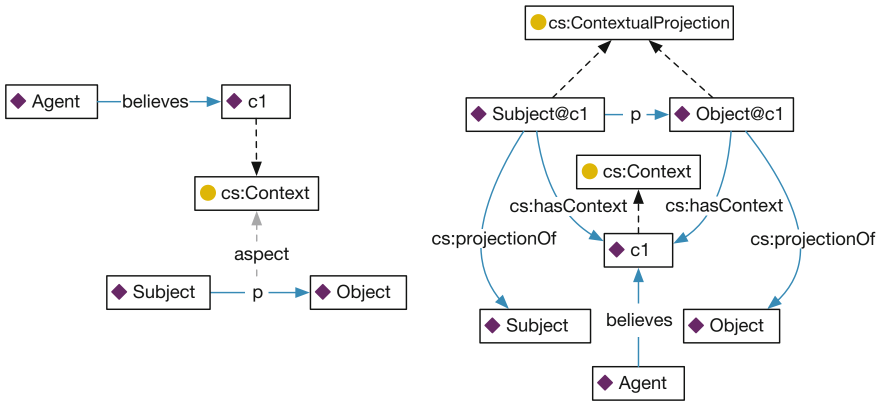


Fig. 6 A transformation by the weaver of a context aspect representing an agent's belief using the context slices pattern

relation between possible worlds. The "actual" world is represented by an individual Agent_i and may be interpreted as the agent believing the axioms associated with it. Each possible world may be interpreted as a context. A fact in the ontology may be associated with a context by connecting it to a class expression. A context individual being of this type may be interpreted as the fact being true in this context. The agent being connected to at least one context where the fact is valid means that the agent beliefs the fact to be true.

Figure 6 shows how the weaver transforms the aspects into an ontology by applying the pattern.

6 Evaluation

The research question tackled in this research is whether aspect-oriented ontology design yields better modules in terms of potential reusability than the recommended ontology design patterns for the modeling problems addressed in this paper.

Research in the field of quality assessment of ontology modules is relatively new. Recent works in the field recommend reusability metrics for ontology modules [14, 15]. Since achieving better reuse of modules is the goal of our research, we use these metrics in order to conduct a comparative evaluation with reuse as the measured parameter.

The above works propose ontology module *cohesion* and ontology module *coupling* as adequate metrics for assessing reusability of ontology modules. These metrics have been inspired by equally named metrics used in Software Engineering, used for measuring the reusability of code modules in a software system.

Cohesion is a measure for the internal relatedness of entities in a module. The more relations between entities in a module exist, the higher the coupling of the module. The idea behind this is that a high degree of interrelatedness of concepts and individuals reflects a high degree of semantic interdependence and thereby a high degree of topical relatedness. Generally, the higher the coupling the better a module is suited for being reused, since it is assumed that a highly cohesive module covers a highly specific domain.

Coupling, in contrast, is a measure for the external, i.e. inter-modular relatedness between entities. The more relations between entities in a module to entities in other modules exist, the higher the coupling value. The intuition here is that strong coupling makes it harder to reuse a module, since due to its many external dependencies, it does not stand on its own as much as a loosely coupled module.

6.1 Experiment

We measured the quality of modules yielded by aspect-oriented ontology development and after weaving using the ontology design patterns described in Sect. 5.4.

We used the cohesion and coupling metrics as described by Oh et al. [14].

The authors define the cohesion for an ontology module M as

$$coh(M) = \begin{cases} \sum_{c_i \in M} \sum_{c_j \in M} \frac{sr(c_i, c_j)}{\frac{|M|(|M|-1)}{2}} & \text{if } |M| > 1 \\ 1 & \text{otherwise} \end{cases}$$

$$sr(c_i, c_j) = \begin{cases} \frac{1}{\text{distance}(c_i, c_j)} & \text{if relations exist between } c_i \text{ and } c_j \\ 0 & \text{otherwise} \end{cases}$$

Coupling of a module M is defined as the ratio of the number of external relationships of entities in M to other modules to the number of all relationships (the sum of internal and external relationships):

$$cpl(M) = \sum_{c_i \in M} \sum_{c_j \in M} rel(c_i, c_j),$$

where $rel(c_i, c_j)$ are either relations between classes, such as SubClassOf, EquivalentClasses or DisjointClasses, relations between individuals, including object property assertions as well as, SameIndividual, DifferentIndividuals and relations between classes and individuals (ClassAssertion).

The authors of [14] distinguish between hierarchical and non-hierarchical coupling. Since no implementation of Oh's or Ensan's metrics are publicly available, we reimplemented the metrics in Java.[20] We implemented both hierarchical and non-hierarchical and the consolidated coupling value, considering both relation types.

[20]source code available at https://github.com/RalphBln/onto-module-metrics.

We extended the metric in order to additionally accomodate the category of aspects. Each relation between an aspect class expression and an axiom is counted as one relationship.

6.2 Results

Table 5 shows the results of the measurements. We measured cohesion (*coh*), hierarchical coupling (*cpl h*), non-hierarchical coupling (*cpl nh*), and consolidated coupling (*cpl*).

The results show that for the majority of the cases, aspect-orientation yields better coupling and cohesion values that following the recommended ontology design patterns.

Table 5 Cohesion and coupling values for modularizations achieved by application of design patterns and aspect-orientation. Coupling values were calculated separately for hierarchical and non-hierarchical relations and both combined

Module	Cohesion	Hierarchical coupling	Non-hierarchical coupling	Coupling (combined)
Contextual projection - pattern				
pattern	0.32	0.0	0.0	0.0
base	0.0	0.0	1.0	1.0
Contextual projection - aspects				
aspect	0.67	0.0	0.0	0.0
base	1.0	0.0	0.5	0.5
View inheritance - patterns				
pattern	0.75	0.33	0.0	0.33
base	0.33	0.67	0.0	0.67
View inheritance - aspects				
aspect	0.33	0.0	0.0	0.0
base	0.75	0.0	1.0	0.33
n-ary - pattern				
pattern	0.4	0.67	0.0	0.33
base	0.33	0.0	1.0	0.67
time	1.0	0.0	1.0	0.5
n-ary - aspects				
aspect	0.83	0.33	0.0	0.33
base	0.67	0.0	0.5	0.33
time	1.0	0.5	0.0	0.5

7 Conclusion and Outlook

In this chapter, we have introduced Aspect-Oriented Ontology Development, a novel approach of modeling context in DL-based ontology languages and to the problem of ontology modularization.

We have defined the model theoretic semantics of our approach in the form of a combination of DL and Kripke interpretations and use the correspondence between DLs and modal logics in order to model aspects in terms of possible worlds in the same language as the object domain (OWL 2).

We have described example use cases for the approach and given an overview of tools that support the development and the use of ontology aspects and an aspect-weaver, which uses ontology design patterns in order to transform aspect-oriented ontologies into standard-compliant ones.

We have evaluated the approach in terms of software engineering based reuse metrics with promising first results.

Future work will consist in the further development of the reasoning services described in Sect. 5.3 and in an extended and comparative evaluation of the usability of the approach and its impact on the quality of ontologies developed with it in terms of reuse.

Acknowledgements This work has been partially supported by the InnoProfile Transfer project "Corporate Smart Content" funded by the German Federal Ministry of Education and Research (BMBF).

References

1. Schäfermeier, R., Paschke, A.: Aspect-oriented ontologies: dynamic modularization using ontological metamodeling. In: Proceedings of the 8th International Conference on Formal Ontology in Information Systems (FOIS 2014), pp. 199 – 212. IOS Press (2014)
2. Group, I.A.W.: IEEE standard 1471–2000. Recommended Practice for Architectural Description of Software-Intensive Systems, IEEE (2000)
3. Parent, C., Spaccapietra, S.: An Overview of Modularity. In: Stuckenschmidt, H., Parent, C., Spaccapietra, S.(eds.) Modular Ontologies. Lecture Notes in Computer Science, vol. 5445, pp. 5–23. Springer Berlin Heidelberg (2009)
4. Filman, R., Friedman, D.: Aspect-Oriented Programming Is Quantification and Obliviousness. Workshop on Advanced Separation of Concerns, OOPSLA (2000)
5. Steimann, F.: Domain Models Are Aspect Free. In: Briand, L., Williams, C. (eds.) Model Driven Engineering Languages and Systems. Lecture Notes in Computer Science, vol. 3713, pp. 171–185. Springer, Berlin (2005)
6. Horrocks, I., Sattler, U.: Ontology reasoning in the shoq(d) description logic. In: Proceedings of the 17th International Joint Conference on Artificial Intelligence IJCAI 2001, 199–204. Morgan Kaufmann 2001
7. 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, New York, NY, USA (2003)

8. Schild, K.: A correspondence theory for terminological logics: preliminary report. In: Mylopoulos J., Reiter R. (eds.) Proceedings of the 12th International Joint Conference on Artificial Intelligence. Sydney, Australia, August 24-30, 1991, pp. 466–471. Morgan Kaufmann (1991)
9. Böhme, S., Lippmann, M.: Decidable Description Logics of Context with Rigid Roles. In: Lutz, C., Ranise, S.(eds.) Frontiers of Combining Systems. Lecture Notes in Computer Science, vol. 9322 pp. 17–32. Springer International Publishing, Berlin (2015). doi:10.1007/978-3-319-24246-0_2
10. Schäfermeier, R., Krus, L., Paschke, A.: An Aspect-Oriented Extension to the OWL API - Specifying and Composing Views of OWL Ontologies using Ontology Aspects and Java Annotations. In: Proceedings of the 7th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management, pp. 187–194 (2015). doi:10.5220/0005591601870194
11. Del Vescovo, C., Klinov, P., Parsia, B., Sattler, U., Schneider, T., Tsarkov, D.: Syntactic vs. Semantic Locality: How Good Is a Cheap Approximation? In: Workshop on Modular Ontologies (WoMO) 2012, pp. 40–50 (2012)
12. Paschke, A., Schäfermeier, R.: Aspect OntoMaven — Aspect-Oriented Ontology Development and Configuration With OntoMaven. In: Abramowicz, W. (ed.) 3rd Workshop on Formal Semantics for the Future Enterprise (FSFE 2015), Business Information Systems Workshops, vol. 228. Springer (2015). arXiv:1507.00212
13. Horridge, M., Parsia, B., Sattler, U.: Laconic and Precise Justifications in OWL. In: Sheth, A., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T., Thirunarayan, K.(eds.) The Semantic Web - ISWC 2008. Lecture Notes in Computer Science, vol. 5318, pp. 323–338. Springer, Berlin (2008)
14. Oh, S., Yeom, H.Y., Ahn, J.: Cohesion and coupling metrics for ontology modules. Inf. Technol. Manag. **12**(2), 81–96 (2011)
15. Ensan, F., Du, W.: A semantic metrics suite for evaluating modular ontologies. Inf. Syst. **38**(5), 745–770 (2013)

Similarity-Based Retrieval and Automatic Adaptation of Semantic Workflows

Ralph Bergmann and Gilbert Müller

Abstract The increasing demand for individual and more flexible process models and workflows asks for new intelligent process-oriented information systems. Such systems should, among other things, support domain experts in the creation and adaptation of process models or workflows. For this purpose, repositories of best practice workflows are an important means as they collect valuable experiential knowledge that can be reused in various ways. In this chapter we present process-oriented case-based reasoning (POCBR) as a method to support the creation and adaptation of workflows based on such knowledge. We provide a general introduction to process-oriented case-based reasoning and present a concise view of the POCBR methods we developed during the past ten years. This includes graph-based representation of semantic workflows, semantic workflow similarity, similarity-based retrieval, and workflow adaptation based on automatically learned adaptation knowledge. Finally, we sketch several application domains such as traditional business processes, social workflows, and cooking workflows.

1 Introduction

Business process management is a well-established discipline that deals with the identification, modeling, analysis, improvement, and implementation of business processes [1]. It is a methodology that is widely applied today to improve the operation of organizations and to align the IT development with business processes. Workflow management is a specific area of business process management that aims at "the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according

R. Bergmann · G. Müller (✉)
Business Information Systems II, University of Trier, 54286 Trier, Germany
e-mail: muellerg@uni-trier.de

R. Bergmann
e-mail: bergmann@uni-trier.de

G.J. Nalepa and J. Baumeister (eds.), *Synergies Between Knowledge Engineering and Software Engineering*, Advances in Intelligent Systems and Computing 626, https://doi.org/10.1007/978-3-319-64161-4_2

to a set of procedural rule" [2]. In the recent years, the use of workflows has significantly expanded from the original domain of business processes towards new areas, which also manifests their relevance in software engineering and development in various ways. For example, in engineering domains, such as software development or chip design, workflows are used to support complex collaborative and distributed development processes [3, 4]. In e-science scientific workflows are executable descriptions of computable scientific processes (i.e. a kind of executable programs) such as computational science simulations and data analyses [5]. Furthermore, workflows can be used to represent and automatically execute search [6] and information integration processes [7] in the context of decision support systems. Even for everyday activities such as cooking, workflows can be used as a means to represent the cooking instructions within a recipe [8] in order to provide step-by-step guidance for the chef.

One of the biggest challenges today arises from the fact that many companies and organizations must be able to more quickly adapt their business according to newly arising market opportunities and demands from the customer, due to actions of the competitors, or due to new technological developments. Agility became an important requirement in may domains in which workflows are applied [1, 9, 10]. Thus, instead of using a small set of standardized workflows, there is an increasing demand for tailoring workflows in a case-specific manner according to the current needs. This asks for intelligent, knowledge-based systems that assist domain experts in the creation or adaptation of workflows. Such systems must be able to represent and reason with knowledge about workflows and workflows elements, such as task, and data items. They must include knowledge that allows to assess the utility of workflows with respect to certain user demands, and they must possess knowledge about appropriate ways to adapt workflows. Consequently, the development of such knowledge-based systems involves a significant knowledge engineering effort that asks for methods from knowledge acquisition, semantic technologies, and machine learning.

In this chapter we present process-oriented case-based reasoning (POCBR) as a method to support the creation and adaptation of workflows. Case-based reasoning (CBR) is an established Artificial Intelligence methodology for experience-based problem-solving by selecting previous problem solutions from the past and adapting them to address a current but related problem [11]. POCBR is a specific sub-branch of CBR that deals with knowledge about processes and workflows [12]. In our own research within the past 10 years, we developed several POCBR methods as well as a generic system called CAKE (Collaborative Agile Knowledge Engine) [13] that support retrieval and adaptation of workflows. Here, experiential knowledge is stored in a repository and consists of semantic workflows, which are best-practices workflows from the past that are semantically annotated using concepts from a domain ontology in order to support their reuse. Users can query the repository with a specification of important properties of the workflow s/he wants to create in order to retrieve potentially reusable workflows. Workflow adaptation methods can then be applied to automatically adapt the retrieved workflow towards the user's query.

In the next section we provide a general introduction into POCBR. Section 3 describes the technical foundations for semantic workflows. The similarity-based retrieval of reusable workflows from a repository is described in Sect. 4, while Sect. 5 presents three different methods for knowledge-based adaptation of workflows. An overview of the CAKE system and selected application examples are described in Sect. 6.

2 Process-Oriented Case-Based Reasoning

Case-based reasoning (CBR) is a problem solving paradigm built upon a rule of thumb suggesting that similar problems tend to have similar solutions [11, 14]. The core of every case-based reasoning system is a case base, which is a collection of memorized experience, called cases. The CBR cycle proposed by Aamodt and Plaza [11] consists of four CBR phases, performed sequentially when a new problem (also called new case or query) must be solved [15]. First, the *retrieve phase* selects one or several cases from the case base with the highest similarity to the query, where similarity is determined by an underlying similarity measure [14]. In the subsequent *reuse phase*, the solutions of the retrieved cases are adapted according to the requirements of the query. In the *revise phase*, the solution determined so far is verified and possibly corrected or improved, e.g., through intervention of a domain expert. Finally, the *retain phase* takes the feedback from the revise phase and updates the knowledge of the CBR problem solver. As part of this learning phase, cases can be added to or deleted from the case base, but also other kinds of knowledge, such as similarity measures or adaptation knowledge can be affected. A unified view on the knowledge contained in a CBR application was proposed by Richter [14] through the metaphor of the four *knowledge containers*: the vocabulary, the case base, the similarity measure, and the adaptation knowledge. The *vocabulary* (which is typically called ontology today) is the basis of all knowledge and experience representation in CBR. The vocabulary defines the information entities and structures (e.g., classes, relations, attributes, data types) that can be used to represent cases, similarity measures, and adaptation knowledge. The *case base* is the primary form of knowledge in CBR, i.e., a repository of cases. A *case* is the representation of a specific experience item (e.g. a problem-solution pair, a problem-solving trace, or a best-practice procedure for performing a certain job) using the predefined vocabulary. The notion of *similarity* plays a key role in CBR, since cases are selected based on their similarity to the current problem. While early CBR approaches were usually restricted to standard similarity measures (such as inverse Euclidean or Hamming distances), the current view is that the similarity measure encodes important knowledge of the domain. Several techniques for *adaptation* in CBR have been proposed so far [15]. However, all adaptation methods require appropriate additional knowledge for adaptation. Motivated by the fact that the manual acquisition of adaptation knowledge is very difficult, several methods have been developed that exploit the knowledge already captured in the cases as source to automatically learn adaptation knowledge

[16, 17]. As a specific approach to knowledge engineering and knowledge-based systems design, CBR is closely related to analogical reasoning, machine learning, information retrieval, databases, semantic web, and knowledge management.

Process-oriented CBR (POCBR) addresses the integration of CBR with process-oriented research areas like Business Process Management and Workflow Management (WFM) [12]. In POCBR a case is usually a workflow or process description expressing procedural experiential knowledge. POCBR aims at providing experience-based support for the automatic extraction [18], design [19], execution [20], monitoring and optimization [21, 22] of workflows. In particular, new workflows can be constructed by reuse of already available workflows that are adapted to new purposes and circumstances. Thereby, the laborious development of workflows from scratch can be avoided.

A case base (or repository) of successful workflows reflecting best-practices in a domain is the core of a POCBR approach. Users can query the repository with a specification of important properties of the workflow s/he wants to create in order to retrieve potentially reusable workflows. One particular characteristic of CBR is that it allows to find cases that do not match exactly the user's query, but which are at least similar in some respect. For example, the CODAW system [23] supports the incremental modeling of workflows by similarity-based reuse of the workflow templates. Leake and Morwick [19] evaluate the execution paths of past workflows in order to support users in workflow modelling by proposing extensions of workflows that are under construction. Besides the retrieval of workflows, also their automatic adaptation is recently addressed in research [24, 25].

3 Semantic Workflows

This section provides a focused introduction to semantic workflows. In particular, we describe our graph-based approach for representation as required for the retrieval phase of POCBR.

3.1 Workflows

A workflow is an executable description of a work process that typically involves several persons and/or resources. It consist of a set of *activities* (also called *tasks*) combined with *control-flow structures* like sequences, parallel (AND) or alternative (XOR) branches, as well as repeated execution (LOOPs). Tasks and control-flow structures form the *control-flow*. In addition, tasks exchange certain *data items*, which can also be of physical matter, depending on the workflow domain. Tasks, data items, and relationships between the two of them form the *data-flow*. Today, various workflow languages are used, depending on the kind of workflow. Languages for business workflows (for example BPMN) have a strong focus on the control-flow, while scientific workflow languages have a stronger focus on the data-flow.

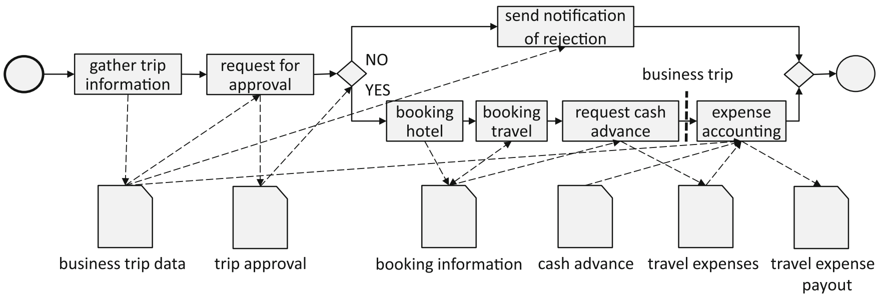


Fig. 1 Examples of a business tip workflow

Figure 1 shows an example of a business workflow. The workflow describes a simple process for a business trip starting with gathering information on the trip and ending with the final expense accounting. It consists of several activities such as booking of the travel or requesting cash advance, which are aligned in sequences (also referred to as control-flow). Furthermore, information is shared between those activities, e.g., business trip data first has to be gathered and then serves as the basics of decision-making whether a trip is approved. Based on the trip approval, the example workflow only executes one specific branch by use of a xor-split/join control-flow pattern.

3.2 *Introduction to Semantic Workflows*

Traditional workflow languages name task and data items by simple textual labels. This makes it difficult to reason with the knowledge captured in such workflows, as required in POCBR. To address this issue, we introduce *semantic workflows* as an approach to cover relevant aspects of the meaning of workflows, including the meaning of the task and data items that occur. Semantic workflows are based on a specifically designed *domain ontology* that consist of sub-ontologies describing the relevant properties of the task and data items that occur in the domain. A traditional workflow is turned into a semantic workflow by adding metadata annotations from the domain ontology to the individual elements of the workflow. The semantic annotations of workflows can then be used as basis for the similarity assessment and adaptation.

The workflow representation in the CAKE system (see Sect. 6) uses an object-oriented representation for ontologies (classes and relations/properties and inheritance) and metadata annotation (linked instances of classes). Thus, tasks and data items can be organized in a hierarchy of classes, in which each item contains certain properties, which can be inherited from the super class. For example, in Fig. 1, the *gather trip information* task includes a property capturing the employee who is assigned to perform this task.

3.3 Representation of Semantic Workflows

In line with previous work on graph-based workflow representation [26, 27], we represent workflows as semantically labeled directed graphs. A *semantic workflow graph* W is a quadruple $W = (N, E, S, T)$ where N is a set of nodes and $E \subseteq N \times N$ is a set of edges. $T : N \cup E \rightarrow \Omega$ associates to each node and each edge a *type* from Ω (see below). $S : N \cup E \rightarrow \Sigma$ associates to each node and each edge a *semantic description* from the domain ontology Σ. A semantic workflow graph contains the following types of nodes (Ω): Each workflow consists of exactly one *workflow node*. Each task in a workflow is represented by a *task node*. Each data item in a workflow is represented by a *data node*. Each control-flow element in a workflow, such as split/join elements for and/xor blocks, are represented as a *control-flow node*. In addition, a semantic workflow graph contains the following types of edges: *part-of edges* for linking the workflow node to every other node, *data-flow edges* which link data nodes to task nodes or vice versa, and *control-flow edges* connecting two task nodes or a task node with a control-flow node. Figure 2 shows the semantic graph representation of the workflow from Fig. 1. It contains one workflow node, which links all elements of the workflow by part-of edges. A simplified fraction of some semantic descriptions are shown in dashed boxes. The business trip data, for example, could include information about the date, venue or expected expenses of the business trip. Further, gather trip information is annotated by information such as the assigned employee or the task's enactment status.

A fraction of the domain ontology Σ for this example domain is illustrated in Fig. 3. The sub-ontology for tasks is shown, which is a light-weight ontology that

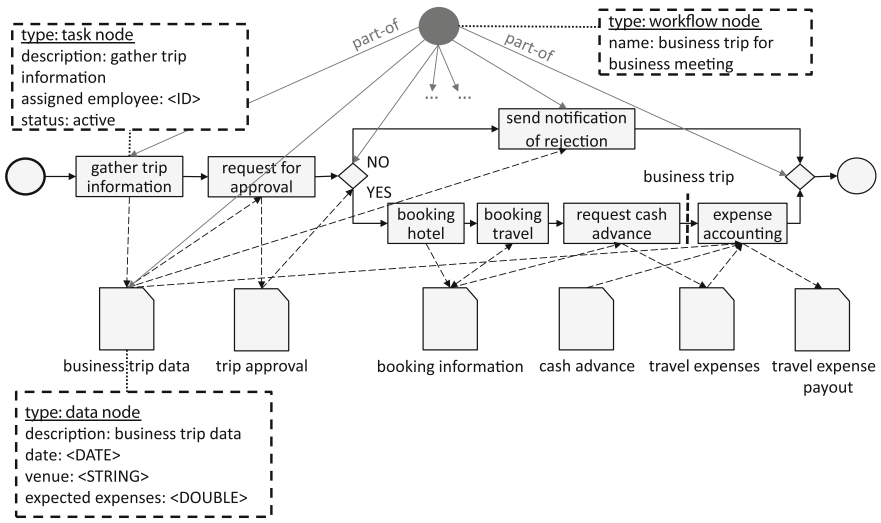


Fig. 2 Semantic workflow of the previously introduced example workflow

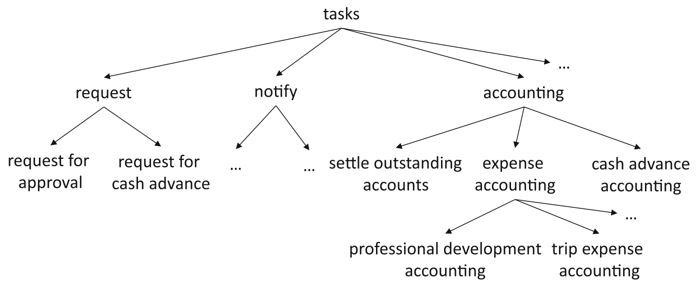


Fig. 3 Example task ontology

arranges relevant tasks in a taxonomical structure. For example, *settle outstanding accounts*, *expense accounting* and *cash advance accounting* are tasks descriptions classified as general *accounting* tasks. Further, *professional development expense accounting* or *trip expense accounting* are special forms of *expense accounting*. In addition, each concept in the ontology may have certain properties (e.g. *assigned employee* in Fig. 2). We employ this ontology for similarity assessment of tasks and data objects (see Sect. 4).

3.4 Repository of Semantic Workflows

A repository of workflows reflecting best-practice processes in a domain is the core of a POCBR approach. In CBR terminology, this repository is the case base that stores the available experience. Given the semantic workflow graph representation, we represent a workflow repository as a set of semantic workflows over the same domain ontology. Hence, a workflow repository is always tied to a particular domain and its semantic metadata representation. Thus, we define a case base $CB = \{CW_1, \dots, CW_n\}$ as a set of cases CW_i each of which is a semantic workflow graph over Σ.

The acquisition of semantic workflows is an important step in the development of a POCBR system. Usually, workflows are already available in a domain (otherwise the use of POCBR is not advisable), but they have to be captured and formalized as part of a knowledge engineering process. For details, see Sect. 6.3.

4 Similarity-Based Retrieval of Workflows

When using POCBR to support the construction of a new workflow, the user has to specify a query stating the requirements on the workflow s/he is aiming at. The CBR approach then selects the most similar semantic workflow from the repository to that query and adapts it in order increase the similarity to the query.

4.1 User Queries

Users can query a repository of semantic workflows with a specification of important properties of the workflow s/he wants to create in order to retrieve potentially reusable workflows. Previous work on workflow reuse and discovery has already addressed the question how typical queries look like. For scientific workflows, Goderis et al. [28] studied the importance of different criteria for workflow discovery. They identified that the task and data items that occur in the workflow are relevant as well as general characteristics of the workflow related to quality, performance, and reliability. The workflow structure is also very important, particularly the data-flow and control-flow as well as the used control-flow constructs. Also for business workflows, the relevant types of queries have been identified and different query languages have been proposed by Beeri et al. [29] and Awad [30]. Their work clearly shows that it is useful to construct queries in the same way as workflows are constructed. Queries can be patterns built from connected workflow elements, which are then matched against the workflows in the repository.

In the light of these results, we focus on queries that are represented as semantic workflow graphs [31]. Such a query specifies some workflow elements together with the semantic description that are considered as requirements on the workflows to be retrieved. A simple query could even consist solely of a workflow node that contains a semantic description specifying the class of workflow and some general properties, such as quality requirements. More sophisticated queries may in addition contain some unconnected data and/or task nodes that specify that these nodes should occur in the workflow one is looking for. Structural properties related to the data and/or control-flow can be specified by linking the nodes with control-flow and/or data-flow links, thus forming a partial workflow (or even several unconnected partial workflows). The aim of retrieval is then to find workflows from the repository that match the specified partial workflows as good as possible according to a similarity measure.

4.2 Semantic Workflow Similarity

The notion of similarity plays a key role in CBR, since cases are selected based on their similarity to the current query. While early CBR approaches were usually restricted to syntactic similarity measures (such as inverse Euclidean or Hamming distances), the current view is that the similarity measure should consider as much semantics as possible. Consequently, similarity measures must be modeled as part of the knowledge acquisition process during CBR application development.

Similarity is formalized as a function that maps a pair of problem descriptions to a real number, often restricted to the unit interval [0, 1], with the convention that a high value indicates a high similarity. Further, similarity is linked with the notions of preference and utility [32–34]: A higher degree of similarity suggests that a case is more useful for solving the current problem and hence should be preferred.

As a means for practical modeling of similarity functions, the so-called *local-global principle*, first proposed by Richter (for details see [15, 32, 33]) is widely used. Modeling similarity means decomposing the similarity function according to certain properties of the case. A *local similarity function* is defined for each individual property, reflecting the utility of a case with respect to the single property only. The local similarities are then aggregated into the *global similarity* by means of an *aggregation function*. This function appropriately combines all local similarity values (e.g. by a weighted average) into a single value that aims at reflecting the utility of the case as a whole. The graph-based representation of workflows introduces structural aspects into the representation and thereby makes the similarity assessment much more complicated. Several graph algorithms have been proposed for similarity assessment such as sub-graph isomorphism, maximal common sub-graphs, or edit-distance measures [27, 35–38].

In our research, we have developed a new similarity model that is an enhancement of the local-global principle [39]. The local similarity measures assess the similarity between two nodes or two edges of the same type based on their semantic description. The global similarity for workflows is obtained by an aggregation function combining the local similarity values within a graph mapping process.

In more detail, the core of the similarity model is a local similarity measure for semantic descriptions $sim_{\Sigma} : \Sigma^2 \rightarrow [0, 1]$. In our example domain the taxonomical structure of the data and task ontology is employed to derive a similarity value that reflects the closeness in the ontology. It is combined with additional similarity measures that consider relevant attributes (see [39] for more details and examples).

The similarity $sim_N : N^2 \rightarrow [0, 1]$ of two nodes and two edges $sim_E : E^2 \rightarrow [0, 1]$ is then defined based on sim_{Σ} applied to their assigned semantic descriptions. The similarity $sim(QW, CW)$ between a query workflow QW and a case workflow CW is defined by means of an admissible mapping $m : N_q \cup E_q \rightarrow N_c \cup E_c$, which is a type-preserving, partial, injective mapping function of the nodes and edges of QW to those of CW. For each query node and edge x mapped by m, the similarity to the respective case node or edge $m(x)$ is computed by $sim_N(x, m(x))$ and $sim_E(x, m(x))$, respectively. The overall workflow similarity with respect to a mapping m, named $sim_m(QW, CW)$ is computed by an aggregation function (e.g. a weighted average) combining the previously computed similarity values. The overall workflow similarity is determined by the best possible mapping m

$$sim(QW, CW) = \max\{sim_m(QW, CW) \,|\, \texttt{admissible map m}\}.$$

Thus, similarity assessment is defined as an optimization problem that consists of finding the best possible alignment of the query workflow with the case workflow. It determines the best possible way (in terms of similarity) in which the query workflow is covered by the case workflow. In particular, the similarity is 1 if the query workflow is exactly included in the case workflow as a subgraph.

This similarity assessment is then used to retrieve the best matching workflow from the repository. While similarity computation by exhaustive search guarantees to find the optimal match, it is computationally not feasible. This is particularly true for retrieval of the best matching workflow in large case bases, since the similarity between the query and each case in the case base must be computed. In our research, we developed four different approaches for an efficient retrieval of workflows, which are briefly summarized below.

4.3 Efficient Similarity Computation by Heuristic Search

In a first step, we improved the efficiency of the similarity computation by developing an A* search algorithm, which is based on a specific well-informed admissible heuristic function [39]. The search algorithm aims at finding an admissible map m between the nodes and edges of the workflows to be compared. In the search space the search nodes represent partial maps, which are incrementally extended towards a complete admissible map. As in traditional A*-search [40], in each search step, the search node with the best (in our case the highest) value for the function $f(node) = g(node) + h(node)$ is selected. Here g represents the similarity value already achieved by search node's mapping and h represents an admissible heuristic function providing a good over-estimation of the additional similarity increment that can be achieved by mapping the workflow elements that are not mapped already. With a memory-bound version of A* we achieved a significant speed-up (up-to several orders of magnitude) in similarity computation over exhaustive search while only slightly compromising the precision of the result.

4.4 Parallelized Similarity Computation

An improved version of the presented A* search algorithm results from parallelizing the similarity computations of several (or all) cases of the case base in order to find the k most similar cases. Therefore the search process is parallelized, maintaining one search queue for each case. In every step, the search node from the queue with the highest f-value from all queues is expanded. Search terminates, when at least k searches have terminated and when the similarity of the k-best case is higher than all f-values of the remaining queues. Since the f-values are upper bounds for the final similarity, it is ensured that none of the remaining cases can ever exceed the similarity of the known k-best case. Hence, completeness of k-best retrieval is guaranteed. This approach can also be executed using parallel threads on multi-core CPUs. In our experiments, this approach again leads to a speed-up compared to the A* up-to an order of magnitude for case bases with a few hundred cases and values of $k < 10$.

4.5 Two-Step Retrieval

If the size of the case base is further increased, additional approaches to speed-up retrieval are required. For this purpose a two-level retrieval method has been developed [41] inspired by the MAC/FAC (Many are called, but few are chosen) model originally proposed by Gentner and Forbus [42]. The first retrieval step (MAC phase) performs a rough and efficient pre-selection of a small subset of cases from a large case base. Then, the second step (FAC phase) is executed to perform the computationally expensive graph-based similarity computation on the pre-selected cases only. This method improves the retrieval performance, if the MAC stage can be performed efficiently and if it results in a sufficiently small number of pre-selected cases. However, there is a risk that the MAC phases introduces retrieval errors, as it might disregard highly similar cases due to its limited assessment of the similarity. Hence, the retrieval approach for the MAC phase must be carefully designed such that it is efficient and sufficiently precise in assessing the similarity. We address this problem by proposing an additional feature-based case representation of workflows, which simplifies the original representation while maintaining the most important properties relevant for similarity assessment. This representation is automatically derived from the original graph-based representation. The MAC stage then selects cases by performing a similarity-based retrieval considering the simplified workflow representation.

4.6 Cluster-Based Retrieval

As alternative approach to efficient retrieval we explored the idea to cluster the workflows using a hierarchical clustering algorithm employing the described similarity measure for assessing the distance of two workflows [43]. Therefore, we developed a hierarchical version of the traditional *Partitioning Around Medoids* (PAM) algorithm [44]. It constructs a cluster-tree where each cluster is represented by a particular case (medoid) such that the case base is partitioned into sets of similar cases. This cluster-tree is then used as an index structure during retrieval.

For retrieving the k most similar cases, clusters at predefined levels in the tree are selected that are most similar to the query. Therefore the similarity between the query and a cluster of cases is computed based on the similarity between the query and the medoid representing the cluster. Only the cases within the selected clusters are then considered for similarity-based retrieval. Our investigation revealed that this approach can decrease the retrieval time without a considerable loss of retrieval quality. Furthermore, parameters enable to control the trade-off between retrieval quality and retrieval time. A significant advantage compared to the MAC/FAC approach is that no additional retrieval phase with a separate simplified feature representation must be designed and thus the development and maintenance effort for retrieval is not increased.

5 Workflow Adaptation and Learning Adaptation Knowledge

As introduced in the previous sections, similarity-based retrieval of semantic workflows for a given user query is an important first step to support the reuse of best-practice workflows from a repository. However, finding a similar workflow does not guarantee that it perfectly matches the query. Several requirements stated in the query (see Sect. 4.1) might not be fulfilled by the most similar workflow. Consequently, more or less comprehensive modifications of the workflow are required. To support the user in performing such modifications, an automated workflow adaptation approach is desirable. The overall aim of automated adaptation is to modify the retrieved workflow in such a way that its original similarity to the query is further increased. Ideally, if the adaptation is able to consider all user requirements perfectly, a similarity of 1 is achieved.

In general, adaptation methods in CBR can be roughly classified into transformational, compositional, and generative adaptation [45]. While transformational adaptation (e.g., [46]) relies on adaptation knowledge represented as rules or operators, generative adaptation demands general domain knowledge appropriate for an automated from scratch problem solver. In compositional adaptation several components from various cases are reused during adaptation, incorporating transformational or generative adaptation methods. Also generalization of cases can be used for the purpose of adaptation, since a single generalized case (e.g., [47, 48]) comprises adaptation knowledge, which provides solutions for a range of problems. Thus, all adaptation approaches require some kind of adaptation knowledge in the particular application domain, for example, in the form of rules describing the replacement of domain-specific tasks or data items. However, the acquisition of such adaptation knowledge is a complex and laborious task. This results in a knowledge-acquisition bottleneck of adaptation knowledge [49], impeding successful workflow adaptation. Thus, various approaches have been proposed to learn adaptation knowledge automatically [16, 50, 51].

In our work, we developed various adaptation approaches for POCBR in order to support individual workflow reuse. As we aim at avoiding the acquisition bottleneck for adaptation knowledge, for each adaptation method described in the following, a specific approach for learning the required adaptation knowledge is presented. This adaptation knowledge is determined prior to retrieval and adaptation from the workflows stored in repository.

5.1 Adaptation by Generalization and Specialization

Generalization and Specialization is an adaptation approach in CBR, in which the adaptation knowledge is learned by a generalization of the cases. The generalized cases are then stored in the case base. Thus, for a particular problem scenario a gener-

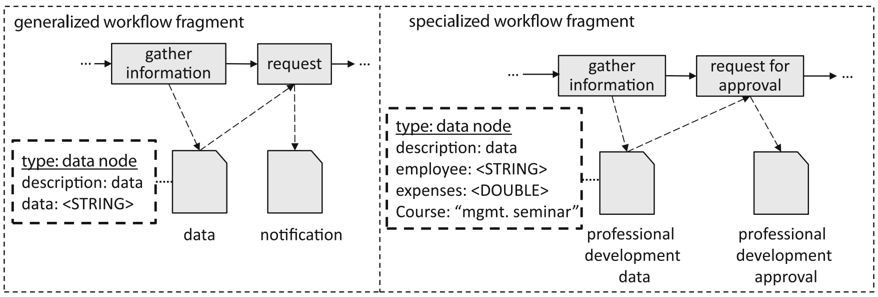


Fig. 4 Example of a generalized and a corresponding specialized workflow

alized case can be retrieved from the case base and refined according to the problem at hand. This approach can be easily applied to workflow cases. A generalized workflow [52] is a workflow whose elements (task and data items) are described by generalized terms from the domain ontology (see Sect. 3). Each generalized term represents multiple specific task or data objects. Thus, a generalized workflow represents several specialized workflows.

An example is sketched in Fig. 4, illustrating a generalized workflow fragment and a corresponding specialized workflow fragment. The generalized fragment can be derived from the example workflow given in Fig. 2. It describes that a request requires the gathering of some data prior to a notification, thereby making no assumption on the concrete request or data present. Thus, it is a generalized process fragment that can be used for many scenarios, for example, the approval for professional development as illustrated in the corresponding specialized workflow fragment. This process fragment involves particular task and data items suited to the particular professional development scenario. A specialized workflow consequently represents the entire process for a concrete scenario. Please note that specialization usually involves information at different levels of abstraction, i.e., professional development data involves information on the particular course, expenses, or related employee, while general data makes no concrete assumption on the information given. Thus, specialization of tasks or data items may also result in a replacement of the entire semantic description (see attached boxes).

A generalized workflow can be learned by comparing similar workflows from the repository (see [52] for technical details). This approach is based on the assumption that if similar workflows contain similar terms, these terms can be replaced by a more generalized term from the ontology. This aims at learning only reasonable generalizations in order to ensure adaptation quality. For example, the generalized workflow data item *data* illustrated in Fig. 4, could result from the fact that similar workflows contain the terms *professional development data, business trip data*, and *business meeting data*. Then the illustrated workflow can be generalized to contain any kind of *data*. Accordingly, the generalized task *request* represents all possible kinds of requests.

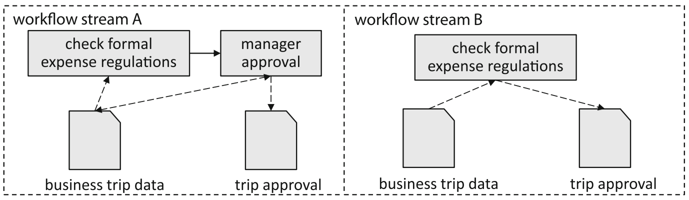


Fig. 5 Example of two workflow streams A and B

Adaptation is supported by specializing a workflow according to the query. This means that each generalized task or data item is replaced by a specialized node such that the similarity between the query and the adapted workflow is maximized. For example, if the generalized workflow contains the term *data* and the query defines that *business meeting data* is desired, then the generalized element description is specialized to *business meeting data*. If a generalized workflow covers several specialized workflows, the workflow repository size can be reduced. This simplifies the repository management and increases the workflow retrieval performance.

5.2 Compositional Adaptation

The idea of compositional adaptation [53] is that each workflow can be decomposed into meaningful sub-workflows. This decomposition is based on the fact that the final workflow output is quite often achieved by producing partial outputs that are somehow combined to create the final workflow output. Partial outputs are generated by particular parts of the workflow (sub-graphs), which we refer to as *workflow streams*. For example, the task *request for approval* illustrated in Fig. 2 could be alternatively performed by the workflow streams illustrated in Fig. 5. These workflow streams describe two alternative decision processes for tip approval. Stream A may refer to a business domain in which a manager approval is required, while stream B could represent a particular process in a government domain as it merely focuses on the formal expense regulations. In general, there can be many workflow streams that could potentially be exchanged with one another to achieve the same partial output.

The basic idea for compositional adaptation is to adapt a workflow by replacing workflow streams in the retrieved workflow with more suitable workflow streams from other workflows. This means that the workflow streams represent the required adaptation knowledge for compositional adaptation. Thus, prior to adaptation, useful workflow streams are extracted from the workflows stored in the repository. Workflow streams can be identified by collecting all data-flow connected tasks[1] until a new data item is created, denoting the corresponding partial workflow output (see for [53] details).

[1]if a tasks consumes a data item produced by another one, they are data-flow connected.

In order to adapt a workflow, a workflow stream can be replaced by a stream learned from another workflow that produces the same partial output but in a different manner, i.e., with other task or data items. Workflow streams can only be replaced, if their data nodes indicate that they represent the same kind of sub-process. This ensures that replacing an arbitrary stream does not violate the correctness of the workflow. In the given example, workflow streams can only be replaced if they consume a data item *business trip data* and produce a *trip approval* data item.

The overall compositional adaptation process aims at increasing the similarity between the query and the adapted workflow by successively replacing workflow streams. Each workflow stream is replaced by the respective stream, which maximizes the overall workflow similarity. The resulting adapted workflow is thus a local optimum achieved by adapting the retrieved most similar workflow using the available workflow streams.

5.3 Transformational Adaptation

Transformational adaptation is based on adaptation knowledge in the form of adaptation rules or adaptation operators [45] that specify a particular modification of the case. Our transformational approach to workflow adaptation focuses on workflow adaptation operators [54] which are specified in a STRIPS-like manner. An operator consists of two workflow sub-graphs we call *streamlets*: a DELETE-streamlet specifies a workflow fraction to be deleted from the workflow and an ADD-streamlet represents a workflow fraction to be added to the workflow. Thereby operators can define the insertion, the removal, or the replacement of a particular workflow fraction. In contrast to compositional adaptation, not only workflow streams can be replaced, but basically any fraction of a workflow, such as a single task or a single data item.

The example adaptation operator shown in Fig. 6 describes the transformation of a planned business trip towards a spontaneous short time customer meeting near-by. Thus, the booking of hotel and travel as well as the request for cash advance becomes superfluous. Instead, just a taxi would have to be ordered. This change does not only affect the activities but also the data items of the workflow, since here the data item *booking information* is no longer required.

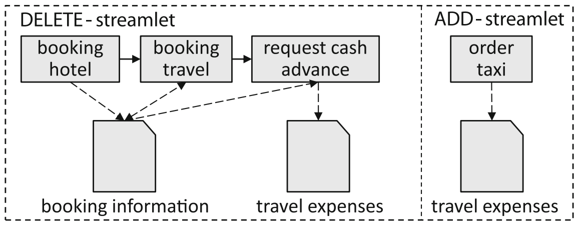


Fig. 6 Example of a workflow adaptation operator

Workflow adaptation operators can be learned from the workflow repository by analyzing pairs of highly similar workflows (selected by using a similarity threshold). For each pair, the difference is determined and workflow operators are generated, whose ADD and DELETE-streamlets basically cover those differences. Roughly speaking, the generated operators thus transform one workflow of the pair into the other one (see [54] for a detailed description of the algorithm).

For adaptation, the learned adaptation operators are applied using a local search algorithm, in a similar manner as in compositional adaptation. The resulting adapted workflow is thus a local optimum achieved by adapting the retrieved most similar workflow using the available adaptation operators.

The three described adaptation methods can also be combined to a single adaptation process (see [55] for details). This comprehensive adaptation, integrates and combines all adaptation methods, thereby maximizing the opportunity to generate a suitable workflow for the given query.

6 CAKE - An Integrated System for Process-Oriented Case-Based Reasoning

In the following, we briefly sketch the CAKE framework, which includes the previously introduced methods and we highlight several application examples.

6.1 Achitecture

The CAKE[2] architecture [13] (see Fig. 7) basically consist of data bases as well as a client and a server component. The latter includes a storage layer which handles persistence of all data, an interface layer for client communication and two central engines, i.e., the agile workflow engine and the knowledge engine working together on the same data items accessed via the storage layer. CAKE is implemented in JAVA as Web-based system running as a Software as a Service.

The *agile workflow engine* is used for the enactment of agile workflows and supports their collaborative modeling. Furthermore, changes on demand can be collaboratively performed on workflow instances as well as workflow models at any time. Running workflow instances delegate tasks to humans via the worklist manager and applications may be invoked via the service connector. In this chapter, however, we mainly focused on the *knowledge engine*, which supports users in finding, defining, and adapting workflows according to their current needs. Therefore, the knowledge engine implements the retrieval and adaptation of semantic workflows as previously introduced. CAKE ensures that any stored resource (a workflow, a task, a document, and any further workflow related resources) is accessible and possesses a clear

[2]See http://cake.wi2.uni-trier.de.

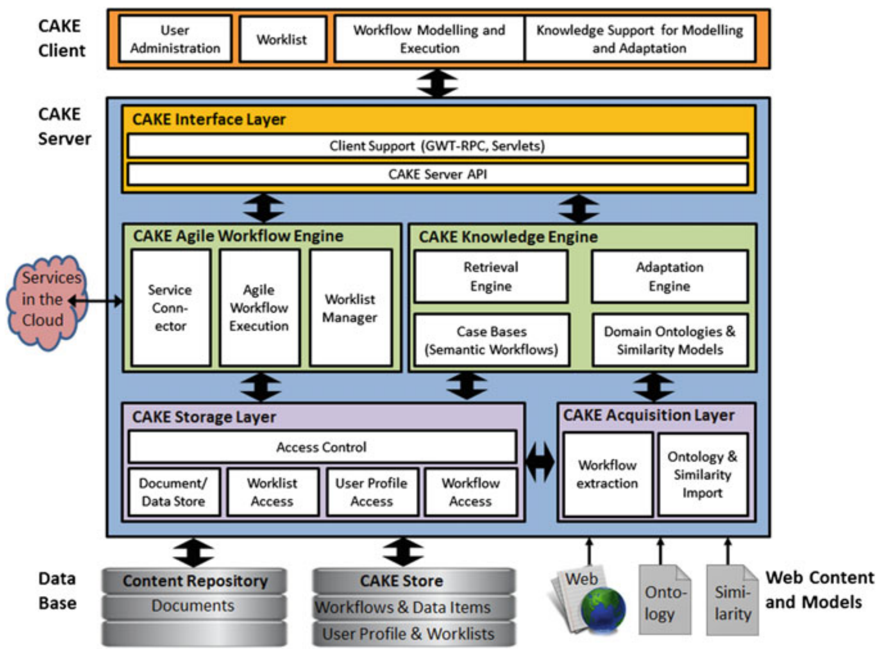


Fig. 7 CAKE system architecture

ownership by means of the resource model implemented in the *storage layer*. This way, workflows can be shared and reused considering particular access rights [56]. The *acquisition layer* handles the import of ontologies and similarity measures and further supports the automatic extraction of workflows from text [57, 58]. CAKE provides two client interfaces, i.e., a browser based access as well as a mobile android application, which are connected to the server component via the *interface layer*.

6.2 Selected Application Examples

The methods for supporting workflow reuse presented in this chapter have potential to be useful in many application areas, some of which will be now briefly sketched.

From the traditional business perspective, the presented methods can support the creation of *business processes* addressing the individual needs of customers or of a particular business scenario. Furthermore, adaptation enables the automatic modification of workflow models, for example, to suit changed business environments. Moreover, POCBR can be employed as a knowledge management method for capturing, storing, and sharing procedural knowledge among different employees or departments.

Social Workflows are a new research area [59, 60] which addresses the support of processes enacted during daily life, such as, do-it-yourself car repair, moving to another city, or organizing a trip with a group of friends. The steps in a social workflow involve access to social networks, the activation of online services, as well as activities performed by several people (e.g., friends or professionals). In a social workflow management system as introduced by Görg [59, 60], the reuse capabilities illustrated in this chapter are highly relevant since the users of social workflows are not experienced in workflow modeling.

In the *cooking domain* a recipe can also be represented as a workflow describing the instructions for cooking a particular dish [55]. While traditional recipe websites solely regard ingredients, categories or recipe names during recipe search, CAKE is able to consider additional knowledge such as required cooking steps, difficulty level, costs, resource consumption, available tools, and diaries. Cooking workflows can be selected and adapted considering particular user preferences by employing the previously introduced POCBR methods within the knowledge engine. Subsequently, CAKE provides a step-by-step guidance for the preparation of the particular dish.

6.3 Required Knowledge Engineering

In order to apply POCBR (including the methods described in this chapter) in a certain domain requires a knowledge engineering process. This knowledge engineering involves the development of the ontology (including task and data sub-ontologies), the similarity measures, as well as the workflows for the case base. The manual acquisition of adaptation knowledge is not required as this knowledge is obtained by the described machine learning approaches particularly targeting the knowledge required for the various adaptation methods.

Ontology development can be performed according to standard methodologies (see, for example [61], Chap. 4 for an introduction). In particular, the reuse of existing ontologies is highly recommended. In the cooking domain, for example, we make use of the cooking ontology developed within the Taaable project[3] [62], which already includes a huge set of ingredients and cooking steps. The sub-ontologies we currently use consist of 156 ingredients and 55 cooking steps.

Based on the ontology, local similarity measures for comparing task and data items must be developed. The knowledge engineering process for similarity measures is well established in CBR [63] and ends-up in selecting appropriate local measures from a similarity-measure library and selecting their parameters according to the domain of the attributes (see, for example [33], Chap. 4).

The final knowledge-engineering steps aims at the acquisition of semantic workflows to populate the case base. For this purpose, different approaches are possible.

[3]http://wikitaaable.loria.fr.

- Workflows can be acquired in a manual process by using workflow modeling tools. Then, appropriate semantic annotations must be added to each task and data item. This manual process is obviously a quite laborious activity, but it ensures a high quality of the resulting knowledge.
- Alternatively, already existing workflows represented in some standard format, such as BPMN, can be reused. For example, existing workflow repositories collected and published for research purposes can be a good starting point. Current collections include the BPM Academic Initiative Model Collection,[4] selected reference models from the IBM Academic Initiative program,[5] the SAP reference models [64], the collections used in the process model matching contests[6] in 2013 and 2015 as well as the collection of the Institute for Information Systems at the DFKI in Saarbrücken [65]. However, the models in these collections lack the required semantic descriptions and thus also requires an additional annotation process before being usable as cases.
- The third approach for acquiring workflows is to apply information extraction methods on available textual descriptions of processes [57, 58, 66]. Cooking recipes, for example, are appropriate textual descriptions. The preparation instructions for the dish included in a recipe can be analyzed and turned into a formal workflow representation. Also in other domains, workflows are described in a textual fashion, for example procedures for technical diagnosis. However, the resulting workflows still need manual quality control and improvement as automatic methods are yet unable to produce results with sufficient quality.

From our experience, the described knowledge-engineering approach is appropriate and the effort is acceptable in many domains. As benefit from these development efforts, POCBR comprehensively supports the creation of new workflows by reuse. Of course, there is no guarantee that workflows that are produced as a result of an adaptation process are always semantically correct (the syntactic correctness, however, is guaranteed). Their correctness depends on the correctness of the learned adaptation knowledge. As this learning process is an inductive process, its correctness cannot be ensured. Thus, there is always a need for a human user to access and validate the resulting adapted workflows.

7 Conclusions

In this chapter, we introduced process-oriented case-based reasoning as a method to support flexible and more individual workflows. We presented an overview of different methods from knowledge representation, knowledge engineering and machine learning to support the representation of semantic workflows as well as their simi-

[4]http://bpmai.org/download/index.html.

[5]https://developer.ibm.com/academic.

[6]https://ai.wu.ac.at/emisa2015/contest.php.

larity based retrieval and adaptation. These methods have been demonstrated using an example from a traditional business process, involving several manual activities typically enacted with support by some specific business application software. Thus, those workflows are a means to coordinate the work among the involved employees but they are also a means for application integration. As pointed out in the introduction, workflows can also used in several other application contexts. Here, the spectrum is quite large, leading applications involving a flow of activities performed completely manual (such as in the cooking domain) up to applications in which the workflows are executed fully automatically (e.g. scientific workflows or workflows for information integration).

Within the scope of our work, we have extensively evaluated the proposed methods, including the quality of the adapted workflows in the cooking domain [39, 52–54]. Initial experimental evaluations in the domain of business processes [67], and social workflows [59, 60] have been performed as well. Future work will focus on investigating these and other new application domains in more depth. Moreover, we will investigate interactive methods, which involve the user during search and adaptation of workflows to further enhance the usability of the presented methods.

Acknowledgements This research was funded in part under grant BE1373/3-1 from the German Research Foundation (DFG) and enacted in cooperation with Mirjam Minor and her group from Goethe University Frankfurt.

References

1. van der Aalst, W.M.: Business process management: a comprehensive survey. ISRN Softw. Eng. **2013**, 1–37 (2013)
2. Workflow Management Coalition: Workflow management coalition glossary & terminology (1999)
3. Freßmann, A., Sauer, T., Bergmann, R.: Collaboration patterns for adaptive software engineering processes. In: Czap, H., Unland, R., Branki, C., Tianfield, H. (eds.) Self-Organization and Autonomic Informatics (I), vol. 135, pp. 304–312. IOS Press, Amsterdam (2005). ISBN 1-58603-577-0
4. Minor, M., Tartakovski, A., Schmalen, D., Bergmann, R.: Agile workflow technology and case-based change reuse for long-term processes. International Journal of Intelligent Information Technologies **4**(1), 80–98 (2008)
5. Taylor, I.J., Deelman, E., Gannon, D.B.: Workflows for e-Science. Springer, Berlin (2007)
6. Freßmann, A.: Adaptive workflow support for search processes within fire service organisations. In: Reddy, S.M. (ed.) Proceedings of the Fifteenth IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, pp. 291–296. IEEE Computer Society (2006)
7. Hung, P., Chiu, D.: Developing workflow-based information integration (WII) with exception support in a web services environment. In: Proceedings of the 37th Annual Hawaii International Conference on System Sciences, 2004, p. 10 (2004)
8. Minor, M., Bergmann, R., Görg, S., Walter, K.: Adaptation of cooking instructions following the workflow paradigm. In: Marling, C. (ed.) ICCBR 2010 Workshop Proceedings (2010)
9. Fleischmann, A., Schmidt, W., Stary, C., Augl, M.: Agiles prozessmanagement mittels subjektorientierung. HMD Praxis der Wirtschaftsinformatik **50**(2), 64–76 (2013)

10. Reichert, M., Weber, B.: Enabling Flexibility in Process-aware Information Systems: Challenges, Methods, Technologies. Springer Science & Business Media, Berlin (2012)
11. Aamodt, A., Plaza, E.: Case-based reasoning: Foundational issues, methodological variations, and system approaches. AI Commun. **7**(1), 39–59 (1994)
12. Minor, M., Montani, S., Recio-García, J.A.: Process-oriented case-based reasoning. Inf. Syst. **40**, 103–105 (2014)
13. Bergmann, R., Gessinger, S., Görg, S., Müller, G.: The collaborative agile knowledge engine CAKE. In: Goggins, S.P., Jahnke, I., McDonald, D.W., Bjørn, P. (eds.) Proceedings of the 18th International Conference on Supporting Group Work, Sanibel Island, FL, USA, November 09–12, 2014, pp. 281–284. ACM (2014)
14. Richter, M.M., Weber, R.O.: Case-Based Reasoning - A Textbook. Springer, Berlin (2013)
15. Lopez De Mantaras, R., McSherry, D., Bridge, D., Leake, D., Smyth, B., Craw, S., Faltings, B., Maher, M.L., Cox, M.T., Forbus, K., Keane, M., Aamodt, A., Watson, I.: Retrieval, reuse, revision and retention in case-based reasoning. Knowl. Eng. Rev. **20**(3), 215–240 (2005)
16. Craw, S., Wiratunga, N., Rowe, R.C.: Learning adaptation knowledge to improve case-based reasoning. Artif. Intell. **170**(16), 1175–1192 (2006)
17. Badra, F., Cordier, A., Lieber, J.: Opportunistic adaptation knowledge discovery. In: McGinty, L., Wilson, D.C. (eds.) Case-Based Reasoning Research and Development, 8th International Conference on Case-Based Reasoning, ICCBR 2009. Lecture Notes in Computer Science, vol. 5650, pp. 60–74. Springer, Berlin (2009)
18. Dufour-Lussier, V., Ber, F.L., Lieber, J., Nauer, E.: Automatic case acquisition from texts for process-oriented case-based reasoning. Inf. Syst. **40**, 153–167 (2014)
19. Leake, D.B., Wilson, D.C.: Combining CBR with interactive knowledge acquisition, manipulation and reuse. In: Proceedings of the Third International Conference on Case-Based Reasoning and Development. ICCBR '99, pp. 203–217. Springer, London (1999)
20. Bergmann, R., Freßmann, A., Maximini, K., Maximini, R., Sauer, T.: Case-based support for collaborative business. In: Proceedings of the 8th European Conference on Advances in Case-Based Reasoning. ECCBR'06, pp. 519–533. Springer, Berlin (2006)
21. Montani, S., Leonardi, G.: Retrieval and clustering for supporting business process adjustment and analysis. Inf. Syst. **40**, 128–141 (2014)
22. Sauer, T., Maximini, K.: Using workflow context for automated enactment state tracking. In: Minor, M. (ed.) Workshop Proceedings: 8th European Conference on Case-Based Reasoning, Workshop: Case-based Reasoning and Context Awareness (CACOA 2006), pp. 300–314. Universität Trier (2006)
23. Madhusudan, T., Zhao, J.L., Marshall, B.: A case-based reasoning framework for workflow model management. Data Knowl. Eng. **50**(1), 87–115 (2004)
24. Müller, R., Greiner, U., Rahm, E.: A$_{\text{gent}}$W$_{\text{ork}}$: a workflow system supporting rule-based workflow adaptation. Data Knowl. Eng. **51**(2), 223–256 (2004). doi:10.1016/j.datak.2004.03.010
25. Weber, B., Wild, W., Breu, R.: CBRFlow: enabling adaptive workflow management through conversational case-based reasoning. In: Funk, P., González-Calero, P.A. (eds.) Advances in Case-Based Reasoning, 7th European Conference, ECCBR 2004, Madrid, Spain, August 30 - September 2, 2004, Proceedings. Lecture Notes in Computer Science, vol. 3155, pp. 434–448. Springer, Berlin (2004). doi:10.1007/978-3-540-28631-8_32
26. Champin, P.A., Solnon, C.: Measuring the similarity of labeled graphs. In: Case-Based Reasoning Research and Development, pp. 1066–1067. Springer, Berlin (2003)
27. Dijkman, R., Dumas, M., Garcia-Banuelos, L.: Graph matching algorithms for business process model similarity search. In: Business Process Management, pp. 48–63 (2009)
28. Goderis, A., Li, P., Goble, C.: Workflow discovery: the problem, a case study from e-science and a graph-based solution. Int. J. Web Serv. Res. **5**(4), 2 (2008)
29. Beeri, C., Eyal, A., Kamenkovich, S., Milo, T.: Querying business processes. In: Proceedings of the 32nd International Conference on Very Large Data Bases, VLDB '06, pp. 343–354. VLDB Endowment (2006)
30. Awad, A.: BPMN-Q: a language to query business processes. In: Reichert, M., Strecker, S., Turowski, K. (eds.) Enterprise Modelling and Information Systems Architectures - Concepts

and Applications. Proceedings of the 2nd International Workshop on Enterprise Modelling and Information Systems Architectures (EMISA'07), St. Goar, Germany, October 8–9, 2007. LNI, vol. 119, pp. 115–128. GI (2007)

31. Müller, G., Bergmann, R.: POQL: a new query language for process-oriented case-based reasoning. In: Proceedings of the LWA 2015 Workshops: KDML, FGWM, IR, and FGDB. CEUR Workshop Proceedings, vol. 1458, pp. 247–255. Trier (2015). http://www.wi2.uni-trier.de/publications/2015_MuellerBergmannLWA.pdf
32. Richter, M.M.: Foundations of similarity and utility. In: Proceedings of the 20th International Florida Artificial Intelligence Research Society Conference (FLAIRS 2007). AAAI Press (2007)
33. Bergmann, R.: Experience Management - Foundations, Development Methodology, and Internet-Based Applications. LNAI, vol. 2432. Springer, Berlin (2002)
34. Bergmann, R., Richter, M.M., Schmitt, S., Stahl, A., Vollrath, I.: Utility-oriented matching: a new research direction for case-based reasoning. In: Schmitt, S., Vollrath, I., Reimer, U. (eds.) 9th German Workshop on Case-Based Reasoning, pp. 264–274 (2001)
35. Kapetanakis, S., Petridis, M., Knight, B., Ma, J., Bacon, L.: A case based reasoning approach for the monitoring of business workflows. In: Bichindaritz, I., Montani, S. (eds.) Case-Based Reasoning. Research and Development, ICCBR 2010, pp. 390–405. Springer (2010)
36. Montani, S., Leonardi, G., Lo Vetere, M.: Case retrieval and clustering for business process monitoring. In: Proceedings of the ICCBR 2011 Workshops, pp. 77–86 (2011)
37. Goderis, A.: Workflow re-use and discovery in bioinformatics. Ph.D. thesis, University of Manchester (2008)
38. Leake, D.B., Kendall-Morwick, J.: Towards case-based support for e-science workflow generation by mining provenance. In: Althoff, K.D., Bergmann, R., Minor, M., Hanft, A. (eds.) Advances in CBR, pp. 269–283 (2008)
39. Bergmann, R., Gil, Y.: Similarity assessment and efficient retrieval of semantic workfows. Inf. Syst. **40**, 115–127 (2014)
40. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach. Prentice Hall, Englewood Cliffs (2010)
41. Bergmann, R., Stromer, A.: MAC/FAC retrieval of semantic workflows. In: Proceedings of the Twenty-Sixth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2013, St. Pete Beach, Florida. May 22–24, 2013. (2013)
42. Gentner, D., Forbus, K.D.: MAC/FAC: a model of similarity-based retrieval. In: Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society. Cognitive Science Society (1991)
43. Müller, G., Bergmann, R.: A cluster-based approach to improve similarity-based retrieval for Process-Oriented Case-Based Reasoning. In: 20th European Conference on Artificial Intelligence (ECAI 2014), pp. 639–644. IOS Press (2014)
44. Kaufman, L., Rousseeuw, P.J.: Finding Groups in Data - An Introduction to Cluster Analysis. Wiley, New York (1990)
45. Wilke, W., Bergmann, R.: Techniques and knowledge used for adaptation during case-based problem solving. In: Pobil, A.P.D., Mira, J., Ali, M. (eds.) 11th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, IEA/AIE-98. Lecture Notes in Computer Science, vol. 1416, pp. 497–506. Springer, Berlin (1998)
46. Minor, M., Bergmann, R., Görg, S., Walter, K.: Towards case-based adaptation of workflows. In: Case-Based Reasoning. Research and Development, pp. 421–435. Springer, Berlin (2010)
47. Maximini, K., Maximini, R., Bergmann, R.: An investigation of generalized cases. In: Ashley, K.D., Bridge, D.G. (eds.) Case-Based Reasoning Research and Development, 5th International Conference on Case-Based Reasoning, ICCBR 2003, Trondheim, Norway, June 23-26, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2689, pp. 261–275. Springer, Berlin (2003). doi:10.1007/3-540-45006-8_22
48. Bergmann, R., Vollrath, I.: Generalized cases: representation and steps towards efficient similarity assessment. In: Burgard, W., Christaller, T., Cremers, A.B. (eds.) KI-99: Advances in

Artificial Intelligence, 23rd Annual German Conference on Artificial Intelligence, Bonn, Germany, September 13–15, 1999, Proceedings. Lecture Notes in Computer Science, vol. 1701, pp. 195–206. Springer, Berlin (1999). doi:10.1007/3-540-48238-5_16

49. Hanney, K., Keane, M.T.: The adaption knowledge bottleneck: how to ease it by learning from cases. In: Leake, D.B., Plaza, E. (eds.) Case-Based Reasoning Research and Development, Second International Conference, ICCBR-97, Providence, Rhode Island, USA, July 25–27, 1997, Proceedings. Lecture Notes in Computer Science, vol. 1266, pp. 359–370. Springer, Berlin (1997)
50. Minor, M., Görg, S.: Acquiring adaptation cases for scientific workflows. In: Case-Based Reasoning. Research and Development, 19th International Conference on Case-Based Reasoning, ICCBR 2011. Lecture Notes in Computer Science, vol. 6880, pp. 166–180. Springer, Berlin (2011)
51. Hanney, K., Keane, M.T.: Learning adaptation rules from a case-base. In: Smith, I.F.C., Faltings, B. (eds.) Advances in Case-Based Reasoning, Third European Workshop, EWCBR-96, Lausanne, Switzerland, November 14–16, 1996, Proceedings. Lecture Notes in Computer Science, vol. 1168, pp. 179–192. Springer, Berlin (1996)
52. Müller, G., Bergmann, R.: Generalization of workflows in process-oriented case-based reasoning. In: Russell, I., Eberle, W. (eds.) Proceedings of the Twenty-Eighth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2015, Hollywood, Florida, May 18–20, 2015, pp. 391–396. AAAI Press (2015)
53. Müller, G., Bergmann, R.: Workflow streams: a means for compositional adaptation in process-oriented CBR. In: Lamontagne, L., Plaza, E. (eds.) Case-Based Reasoning Research and Development - 22nd International Conference, ICCBR 2014, Cork, Ireland, September 29, 2014–October 1, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8765, pp. 315–329. Springer, Berlin (2014)
54. Müller, G., Bergmann, R.: Learning and applying adaptation operators in process-oriented case-based reasoning. In: Hüllermeier, E., Minor, M. (eds.) Case-Based Reasoning Research and Development, ICCBR 2015, Frankfurt am Main, Germany, September 28–30, 2015, Proceedings. LNCS, vol. 9343, pp. 259–274. Springer, Berlin (2015)
55. Müller, G., Bergmann, R.: CookingCAKE: a framework for the adaptation of cooking recipes represented as workflows. In: Kendall-Morwick, J. (ed.) Workshop Proceedings from The Twenty-Third International Conference on Case-Based Reasoning (ICCBR 2015), Frankfurt, Germany, September 28–30, 2015. *CEUR Workshop Proceedings*, vol. 1520, pp. 221–232. CEUR-WS.org (2015). http://ceur-ws.org/Vol-1520/paper23.pdf
56. Görg, S., Bergmann, R., Gessinger, S., Minor, M.: A resource model for cloud-based workflow management systems - enabling access control, collaboration and reuse. In: Desprez, F., Ferguson, D., Hadar, E., Leymann, F., Jarke, M., Helfert, M. (eds.) CLOSER 2013 - Proceedings of the 3rd International Conference on Cloud Computing and Services Science, Aachen, Germany, 8–10 May, 2013, pp. 263–272. SciTePress (2013)
57. Schumacher, P.: Workflow Extraction from Textual Process Descriptions. Verlag Dr. Hut, München (2016)
58. Schumacher, P., Minor, M., Walter, K., Bergmann, R.: Extraction of procedural knowledge from the web. In: Workshop Proceedings: WWW'12. Lyon, France (2012)
59. Görg, M.S.: Social Workflows, pp. 77–110. Springer Fachmedien Wiesbaden, Wiesbaden (2016)
60. Görg, S., Bergmann, R.: Social workflows vision and potential study. Inf. Syst. **50**, 1–19 (2015)
61. Hitzler, P., Krötzsch, M., Rudolph, S.: Foundations of Semantic Web Technologies. Chapman and Hall/CRC Press (2010). http://www.semantic-web-book.org/
62. Cordier, A., Lieber, J., Molli, P., Nauer, E., Skaf-Molli, H., Toussaint, Y.: WIKITAAABLE: a semantic wiki as a blackboard for a textual case-base reasoning system. In: 4th Semantic Wiki Workshop (SemWiki 2009) at the 6th European Semantic Web Conference (ESWC 2009), Hersonissos, Greece, June 1st, 2009. Proceedings. CEUR Workshop Proceedings, vol. 464. CEUR-WS.org (2009). http://ceur-ws.org/Vol-464

63. Stahl, A.: Defining similarity measures: Top-down vs. bottom-up. In: Craw, S., Preece, A.D. (eds.) Advances in Case-Based Reasoning, 6th European Conference, ECCBR 2002 Aberdeen, Scotland, UK, September 4–7, 2002, Proceedings. Lecture Notes in Computer Science, vol. 2416, pp. 406–420. Springer, Berlin (2002)
64. Curran, T.A., Ladd, A.: SAP R/3 Business Blueprint: Understanding Enterprise Supply Chain Management. Prentice Hall International, Englewood Cliffs (1999)
65. Thaler, T., Dadashnia, S., Sonntag, A., Fettke, P., Loos, P.: The IWi process model corpus. Technical report, Saarländische Universitäts- und Landesbibliothek, Postfach 151141, 66041 Saarbrücken (2015). http://scidok.sulb.uni-saarland.de/volltexte/2015/6267
66. Schumacher, P., Minor, M.: Extracting control-flow from text. In: Proceedings of the 2014 IEEE 15th International Conference on Information Reuse and Integration, pp. 203–210. IEEE, San Francisco, California, USA (2014)
67. Pfister, M., Fuchs, F., Bergmann, R.: Ähnlichkeitsbasiertes Retrieval von BPMN-2.0-Modellen. In: Lernen, Wissen, Daten, Analysen (LWDA 2016) (2016). http://www.wi2.uni-trier.de/publications/2016_PfisterFuchsBergmann_LWDA.pdf

Development of Knowledge-Based Systems Which Use Bayesian Networks

Isabel M. del Águila and José del Sagrado

Abstract Bayesian networks allow for a concise graphical representation of decision makers' knowledge on an uncertain domain. However, there are no well-defined methodologies showing how to use a Bayesian network as the core of a knowledge-based system, even less if not all the features should be supported by the knowledge model. That is to say, the software, that has to be released to customers, has also to embed functionalities not based on knowledge, concerning to the information management processes closer to the world of a classical software development projects. These components of the software application have to be built according to practices and methods of Software Engineering discipline. This chapter is conceived as a guideline about how to manage and intertwine languages and techniques related to Knowledge Engineering and Software Engineering in order to build a knowledge based system supported by Bayesian networks.

Keywords Knowledge-based systems development · Bayesian networks · Model driven software development

1 Introduction

This chapter addresses, from a practical point of view, how to manage and intertwine languages and techniques related to Knowledge Engineering (KE) and Software Engineering (SE) in order to build a knowledge-based system (KBS) which uses Bayesian networks (BNs). Within the Artificial Intelligence (AI) techniques we focused on BNs because they allow for a concise graphical representation of the

I.M. del Águila (✉) · J. del Sagrado
University of Almería, Crta de la Playa, 04120 Almería, Spain
e-mail: imaguila@ual.es

J. del Sagrado
e-mail: jsagrado@ual.es

G.J. Nalepa and J. Baumeister (eds.), *Synergies Between Knowledge Engineering and Software Engineering*, Advances in Intelligent Systems and Computing 626, https://doi.org/10.1007/978-3-319-64161-4_3

decision makers knowledge on an uncertain domain [40]. However, the methodology applied can also be extended to other KE models by defining the appropriate models and transformations.

Computer applications assist in most of the processes and tasks performed in our society, such as scientific research, engineering problems or decision-making in business. Sometimes software is the keystone that keeps standing a product or service. Nevertheless, the complexity of several domains entails the necessity of using techniques that exceed the typical software development processes. For instance, when heuristic techniques, non-transactional algorithms or probabilistic reasoning are needed to solve a problem, we must use a KE methodological approach for building the features that involve expert knowledge.

The classic KE approaches to system modeling use mostly logic-based symbolic knowledge representation methods. These methods are specific from AI and are usually far from the SE perspective. The development of KBS is a modelling activity which requires a methodology that ensures well-defined knowledge-models (e.g. a Bayesian network) that are able to manage the complexity of the symbol-level in the construction process [44]. However, BNs haven't been addressed in depth from the KE point of view (i.e. with the goal of getting a software application based on a BN), since most of the effort had focused on the study of the concepts and inference processes of the BN itself. Nevertheless, a well-defined knowledge model does not ensure the success of the software released to the customer, because requirements and technology issues are usually divorced from KBS development methods. Three sides of the software product have to be taken into account [11]: the customer role, knowledge engineer role and software engineer role.

This scenario forces us to combine modern approaches of KE, specifically BN modeling, and SE for their integration under the umbrella of techniques and methods applied to both engineerings. This combined approach would allow the development of quality software products using SE and/or KE methods, since there are many cases in which companies must deploy software systems that transparently integrate probabilistic knowledge-based and not-knowledge-based components [10, 15].

In KBS development the knowledge and software modeling phases are not integrable effortlessly due to the different languages needed and the different steps followed in the project. Besides, the knowledge model (i.e. BN) is described in the knowledge level [36] (closer to the world) where we only need to specify what the agent knows and what its goals are, whereas the software models are in the symbol level that is closer to the system. This work focuses on bring closer these two disciplines by applying model-driven software development as an effective approach to deal with the development of knowledge-based systems faster and more customized than classical approaches [12].

First, we will summarize the background of our proposal including an overview of the main KE methodologies, the theoretical probabilistic concepts and the fundamentals of model driven development approach. Next, we will focus on the description of a metamodel which contains the key concepts used in the definition of a BN-based knowledge model. This metamodel will be the base to define a UML extension using profiles that can bridge the gap in representation and facilitate the seamless incorporation of a BN-based knowledge model. In the context of knowledge-based software development, it helps us to build a software system that needs to combine knowledge and not knowledge based components, defining an anchor point between KE and SE. We will also describe a case study in order to guide developers to built a KBS which uses BN as knowledge model.

2 Knowledge Engineering: Current State

The Figure 1 shows a review of the historical evolution of KE intertwining it with the history of SE, because both had parallel and divergent evolutions but following a similar pattern [13]. We start the timeline in 1956 because it is generally thought that is the time where General Motors produced the first operating system. We also describe a set of milestones that represents a convergence or divergence of development methodologies that did not appear at the same time in SE and KE. The two disciplines propose the development of software using engineering principles, so should be similarities between techniques, methods, and tools applied in both fields (see Fig. 1). However, KE and SE have ignored each other, which is against some basic principles of any engineering such as cooperation, reuse or work partition.

The goal of KE is : "... constructing Knowledge Based System (KBS) in a systematic and controllable manner. This requires an analysis of the building and maintenance process itself and the development of appropriate methods, languages and tools suitable for developing KBSs" [44]. We summarize KE evolution in Table 1.

In the first stage (Mastering Intelligence (1956–1978)) Knowledge Engineering had not yet appeared. The term Artificial Intelligence was coined those years. Previously, some authors such as Alan Turing (from 1900 to 1956) had made proposals close to AI meaning. Most works in this period focused on the development of general problem-solving techniques, such as the STRIPS (Stanford Research Institute Problem Solver System) planning system [21] or GPS (General Problem Solver) [20]. However, these techniques were not appropriated for concrete problems, since domain knowledge were required instead of general knowledge. That is, techniques were needed for transferring knowledge from the experts to computers. This last point of view gave birth to the first KBSs, such as PROSPECTOR [26] and MYCIN [42] but without the support of any development methodology.

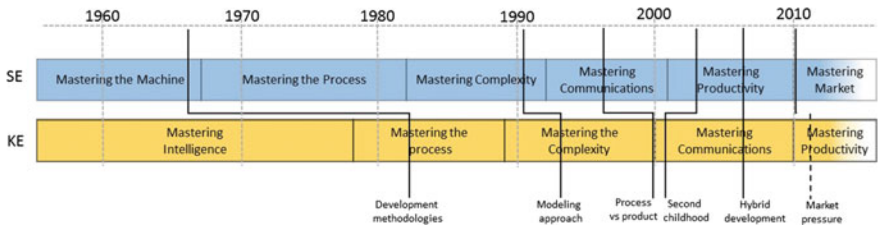


Fig. 1 Timeline of Software Engineering and Knowledge Engineering

Table 1 Knowledge Engineering periods

Era	Periods	Description
Mastering Intelligence (1956–1978)	General solvers	
	KBS	Knowledge
Mastering Process (1979–1989)	Process	Crisis. Knowledge Eng. Shells
	Specialization	Domain specific applications
Mastering Complexity (1990–2000)	Second generation	KBS transfer. Knowledge industry
	Reusing	Tasks libraries. Know. management
Mastering Communications (2001–2010)	Distributed	Ontologies. Semantic Web
	Data Mining	Databases. Automatic learning
Mastering Productivity (2010–2017)	Expanding	Transfer to many domains
	Integration	Integrated approach

As there was no engineering method, the development process had to face many problems. The KBS migration from research project to commercial products was not successful in most cases. This was the starting point of a the second stage, Mastering the process (1979–1989). Products did not meet the customers' expectations, time and cost estimates were not appropriated, and maintenance and testing tasks became costly. Building a KBS was basically conceived as a process of transferring human expert knowledge to a computer knowledge base [45]. This fact caused that knowledge acquisition became the most important task and also the main bottleneck. In the same way the software crisis resulted in establishing SE as a discipline. Early KBSs development problems forced developers to search for methodological approaches and a better analysis of the applied process [3]: Knowledge Engineering was born. Besides, the wide scope of applicability of Artificial Intelligence techniques drove this discipline to specialization and diversification into new disciplines such as computer vision, data mining and pattern recognition.

In the early nineties, with the purpose of overcome the knowledge acquisition bottleneck, a new generation of methodologies redefined KE from transfer/mining to a modeling perspective, (Mastering the Complexity (1990–2000)). This approach is based on the knowledge level concept proposed by Nevell [36]. Starting from this idea, a second-generation of KE methodologies came to light. They were the first attempts to provide a complete methodology for the entire KBS development life cycle. Moreover, the need to enhance productivity led to the empowerment of knowledge component reuse, in the same way that classes and objects are reused in object-oriented development. The concept of distributed software was also extended to KBS, making it possible to apply this technology to a wider range of domains and more complex problems, (Mastering Communications (2001–2010)). The spread of the World Wide Web (WWW) made the ability to manage the semantics of the data which is of paramount importance for the successful discovery, distribution and organization of this information. Two connected disciplines emerged to help coping these problems, Ontological Engineering [24] and Semantic Web [2].

The last stage in KE history is mastering the Productivity (2010- —). Important subjects that have to be taken into account are those related to make AI techniques commercially viable and to extend them to the consumer products (such as interactive smart toys), and those associated to their application to specific domains, such as SE (i.e., Search Based Software Engineering) (SBSE) [25]. Moreover, the need for software systems capable of coordinating information and of managing knowledge in a single product was evident. The development of this kind of software should be approached from a coordinated application of KE and SE methodologies [10, 13] taking advantage of the successful application of methods such as model driven engineering or agilism.

Once the main milestones in KE history have been described, we will keep an eye on how SE methods have been applied in KBS projects, focusing on model driven approaches.

3 Software Engineering Methods for KBS: Model Driven Development

A KBS can be defined as "software that has some knowledge or expertise about specific, narrow domain, and is implemented such that knowledge base and the control architecture are separated. Knowledge-Based Systems have capabilities that often include inferential processing (as opposed to algorithmic processing), explaining rationale to users an generating non-unique results" [33]. The main role is played by the knowledge model (i.e. knowledge base). In general, a model provides an abstraction about reality and the knowledge models abstract the human experts' problem solving approaches in order to be used in the development of software solutions. Knowledge models usually are described in a specific purpose language. For instance, a BN allows to represent uncertainty using probabilities. There is not a standard language, because it heavily depends on knowledge representation mechanisms (e.g. rules, semantic networks, frames, BNs). Contrarily, the not knowledge based components of the application are described using modeling languages from the SE domain (UML is the most used standard). From a commercial point of view, the development of a software application should focus on software features, some of them supported by a knowledge model and others supported by some kind of algorithmic methods. That is, several kind of languages have to be mandatory used and combined in the same development project.

Figure 2 shows how customers, software engineers and knowledge engineers perceive the software development project in which they are involved. The knowledge engineer uses KE methods to carry out the tasks, relegating to the background the activities related to SE issues, such as reporting or interface management. The software product that results from this process is a typical KBS. The software engineer systematically applies its skills, tools and SE methods to develop a software product (system), where knowledge is only another element, and usually forgets to take into

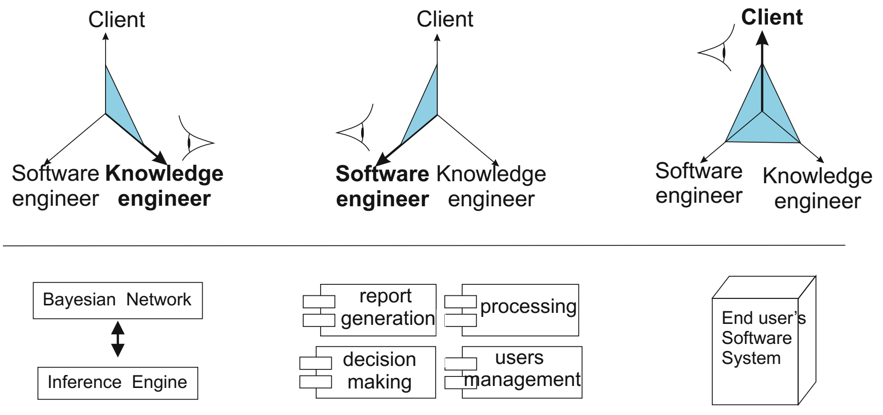


Fig. 2 Different perspectives of a software development project

account the different nature that the knowledge has. Finally, the customer's view of the project focuses on quality and the need of cooperation between the two engineering and their own modeling languages [10, 11], so that the software product to be released properly covers all her/his needs as a black box. This also means that software components based and not based on knowledge must be transparently interwoven.

BNs have its own algebraic notation. If we want a BN to be part of a software solution, we must be able to express itself in the same, or at least in a compatible, language that is commonly used to model general purpose software (i.e. UML).

The use of different modeling languages limits the applicability of one of the software development schemes more widespread in our days, Model-Driven Architecture (MDA) [37]. Model-Driven Engineering (MDE) [41] refers to this software development approach. It uses models as relevant entities, enabling the definition and automatic execution of transformations between models, and from models to code. MDE starts with the well-known and long established idea of separating the specification of the system operation, from the details of the way that system uses the capabilities of its platform. Similarly, KE separates the knowledge base from the inference engine. This approach to system development, which increases the power of models, provides a means for using models in order to direct the course of understanding, design, construction, deployment, operation, maintenance and modification of the software project.

The first type of MDE models is called a Platform Independent Model (PIM). This is a formal specification of the structure and functions of a system that abstracts away technical details. A Platform Specific Model (PSM) specifies the realization of the functionality described in the PIM on a specific platform. A PIM could be transformed into one or more PSMs, applying model-to-model (M2M) transformations. A PSM specifies the system in terms of the implementation components that are available in a specific implementation technology. The final translation in this

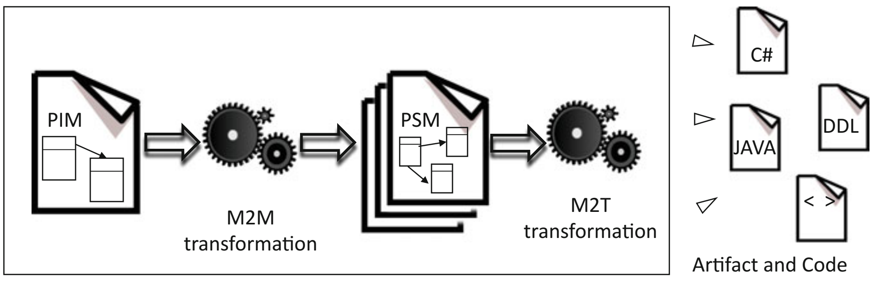


Fig. 3 Model Driven Engineering transformation chain

development approach is to evolve the PSM into code by applying model-to-text (M2T) transformations. Models and model transformations define a transformation chain, as is shown in Fig. 3. Models are specified in a modeling language which is defined by a metamodel.

This approach has as big advantage that there are supporting tools that enable the specification and automatic execution of these transformations (e.g. from UML models to Java code). Once the modeling languages are selected and the transformation chain is made up, the whole effort concentrates solely on creating and reviewing the PIM, because is the chain starting point. A supporting MDE tool will automatically generate the final code.

MDE has been successfully applied to many domains, such as web engineering [34], ontology engineering [22], user interface development [27], rule based system development [6, 7], and a variety of industrial domains [17, 39].

4 Bayesian Networks Basics

A Bayesian network is a type of probabilistic graphical model [31] that uses a directed acyclic graph (DAG) to represent a set of random variables and their conditional dependences. Variables are represented as nodes in the DAG, whereas arcs represent conditional dependences. Each node has associated a probability function, $P(X_i|pa(X_i))$, that takes (as input) a particular assignment of values for the node's parent variables, $pa(X_i)$, and gives (as output) the probability of the variable X_i represented by the node. The joint probability distribution over the set of random variables can be recovered from the set of probability functions as:

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i \mid pa(X_i)). \tag{1}$$

Bayesian networks [28, 30, 40] allow us to represent graphically and concisely knowledge about an uncertain domain. The uncertainty is represented in terms of a probability distribution whose relations of independence are codified into a DAG structure. Bayesian networks also offer an inference mechanism, i.e. propagation algorithms, making possible reasoning under uncertainty. From a practical point of view there are many software platforms for constructing and using probabilistic graphical models. In this work we use, one of them, the Elvira system [19], which is a Java tool to build probabilistic decision systems, because it incorporates a graphical user interface so that BN models can be easily edited in the ASCII format also provided.

4.1 Steps to Build a Bayesian Network

Bayesian networks can be used as knowledge models to represent expert knowledge on an uncertain domain, but their successful application is not easy. The process for obtaining the model can be carried out either *manually*, from expert knowledge on the domain, or *automatically*, from databases. In the first case, the probability elicitation constitutes a bottleneck in the BNs development [18] what is going to be a very difficult task for those who are untrained. In the second case, mistakes can arise due to noise, discretization methods or database errors.

Several authors have defined the process of constructing Bayesian networks (BNs) by establishing the steps to follow to build the knowledge model [32, 35]. These steps are:

0. Feasibility study. A good starting point is to decide if a Bayesian network model is the appropriate one for the type of application or problem. A Bayesian network would not be appropriate [32] when there are no domain experts or useful data for finding knowledge about the domain or if the problem is highly complex, has very limited utility or can be solved by learning an explicit function.
1. Variable identification. At this stage, some modelling decisions have to be made about how to represent the knowledge domain and what is going to be represented in the Bayesian network. The variables involved in the problem, and their possible values or states have to be identified. When the Bayesian network model is being organised, it must not be forgotten that its purpose is to estimate certainties for unobservable (or observable at an unacceptable cost) events [28]. These events are called hypothesis events, and once they are detected, they are grouped into sets of mutually exclusive and exhaustive events to form hypothesis variables. Variable identification can be accomplished either by means of expert elicitation, or applying unsupervised classification (see, for example, [29]).
2. Structure elicitation. The topology of the network must capture relationships between variables (i.e. two variables should be connected by an arc if one affects the other). In general, any independence suggested by a lack of an arc should correspond to real independence in the knowledge domain. However, in the real world, such links cannot be expected to be accomplished easily. Causal relations are not always obvious. When there are data available, we can use structural learn-

ing approaches, such as the work of [43] based on the detection of independences, and that of [9], based on applying search plus score techniques, instead of relying completely on the expert's opinion.

3. Parameter elicitation. This step deals with the acquisition of the numbers necessary to estimate the conditional probabilities for the Bayesian network model. Statistical methods for learning parameters in Bayesian networks can be found in [4, 8]. It is worth to note that structure and parameter elicitation are intrinsically related. In order to estimate the conditional probabilities, the graphic structure have to be known, and some probabilistic independence test will have to be performed to determine this structure and to help establishing the links direction between variables. When enough data are available, structure identification and parameter elicitation can also be done automatically by automatic learning methods [35].
4. Validation and testing. Checks if the model meets the criteria for use and its suitability for the job it is designed to do. In order to validate and understand how to make the best of the network, the process tries to give an answer to questions such as [32]: Does the structure reflect the fundamental independence relationships of the problem? What is the level of predictive accuracy acquired? And, is the network sensitive to changes? Besides, it is important do not forget to measure the usability and performance to find out whether the Bayesian network model meets customer use criteria [32].

Once the BN model is built (i.e. we know the variables and the relationships between them), it can be used to answer probabilistic queries about the variables.

4.2 Reasoning with Bayesian Networks

Reasoning with BNs consists in performing probabilistic inference using the Bayes' theorem to update the knowledge of the state of a subset of variables (of interest) when other variables (the evidence variables) are observed. This process computes the posterior distribution of the unobserved variables given the evidence (i.e. the observed ones). In this way BNs can be used in *diagnostic*. For this, some variables that represent effects receive evidence and the posterior distribution of the cause variable is computed. Also, BNs can be used in *prediction*. Simply one of the variables is considered as the class and the other variables as features predicting that describe the object that has to be classified. Then, posterior probability of the class is computed given the features observed. The value assigned to the class is that it reaches the highest posterior probability value. A predictor based on a BN model provides more benefits, in terms of decision support, than traditional predictors, because it can perform powerful what-if problem analyses.

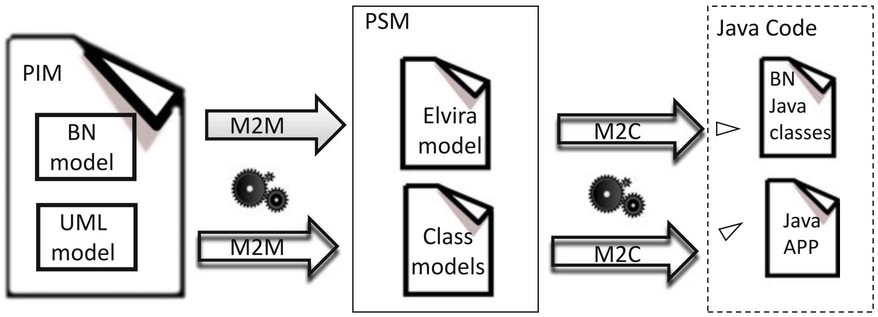


Fig. 4 MDA for BN-based KBS

5 Embedding BNs in MDE Chain

According to MDE notation, a BN is a PIM, Elvira [19] is the platform model selected into which a BN has to be translated in order to get the final PSM (M2M translation). This software platform also offers Java support for the BN, through its application programming interface (API). That is to say, it provides the last link of the MDE approach (i.e. the code) using a M2T transformation (see Fig. 3). Regularly, compatible representations of PSM and PIM are needed when deploying applications that transparently integrate BN-based and not-knowledge-based components. This fact forces developers to manage models that have different nature and notation.

The translation model proposed is shown in Fig. 4. Both, the not knowledge-based PSM and BN PSM need to be expressed in terms of object-oriented design languages coming from SE (i.e. UML) and their translation into code have already been solved by means of a M2C translations. This feature is actually included in many modeling tools, such as Visual Paradigm, Microsoft Visual Studio or Enterprise Architect. This M2C translation for BN models can be achieved by means of the Java API provided by Elvira platform. Nonetheless, BNs lack a modeling language compatible with UML that allows the full application of MDE, contrary to the case of some other representations for knowledge models such as production rules [1, 7]. Thus, expressing in a compatible way the BN and UML models facilitates the inclusion of the single extended PIM in the MDE chain (i.e. the shadow arrow M2M in Fig. 4 has to be defined, as well). A unified language allows a great level of abstraction and makes easy the implementation process. For instance, in case of rules-based models the translation of the rules of PIM models into rules-based web-systems is solved by combining Java Server Faces with Jess rule engine [6].

This integration starts by defining the metamodel for the platform and the domain of the application that is going to be modeled. In our case, this corresponds to the BN metamodel, which has been called *BayNet*. It allows us to specify how BN concepts are related to and represented in standard UML. A metamodel includes domain entities, restrictions between them and limitations on the use of entities and relationships.

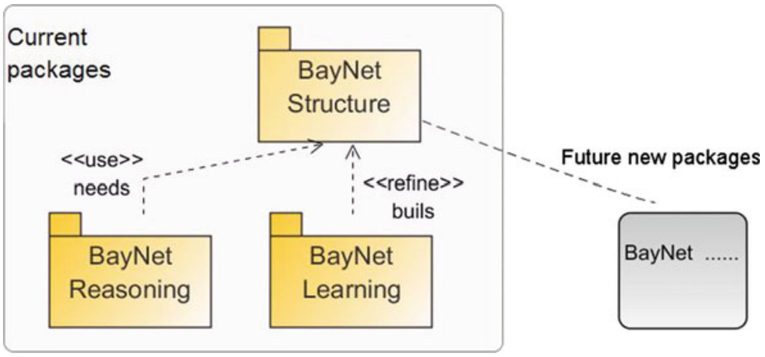


Fig. 5 BayNet metamodel basic structure

5.1 Bayesian Network Metamodel

The *BayNet* metamodel showed in Fig. 5, due to BN complexity, has been split in packages in order to facilitate the evolution of the proposal. The metamodel core (see Fig. 6) consists of *BayNet structure* (as we need to define the BN model) and *BayNet reasoning* (as we need to reasoning with a BN) packages. The *BayNet structure* package represents the basic components of a BN (*Bnet* class): its qualitative (i.e. DAG) and quantitative (i.e. set of probability distributions) parts. The qualitative part is represented by the class *Variable* and its self-association. An *Assignment* consists in assigning a *State* to a *Variable* modifying, accordingly, its marginal probability. The quantitative part is represented by means of the classes *Configuration* and *Relation*. For each given child-father association (*Configuration*) in the directed acyclic graph is assigned a conditional probability value (*Relation*). That is, we assign a probability value to each combination of values of a variable X_i and its parents $pa(X_i)$ in the graph, to define the conditional probability distribution $P(X_i|pa(X_i))$.

In *BayNet* reasoning, an inference can be viewed as a process (*I_Process*), a class able to carry out inferences (*I_Entity*) or an operation inside a class (*I_Task*). These three views allow to model different levels of abstraction in the decision tasks associated to a BN-based KBS. An *Inference* is an aggregation of the observed variables (*Evidence*) together with the execution of the operations needed to compute the posterior probability distributions of the variables of interest (*Propagation*).

5.2 UBN Profile

Once the metamodel is built we have to extend UML to manage the unified PIM model, creating a UML-compatible modeling language for BNs. We can extend UML without changing its semantics by only particularizing some concepts using a series of mechanisms offered by the language itself: the profiles [23].

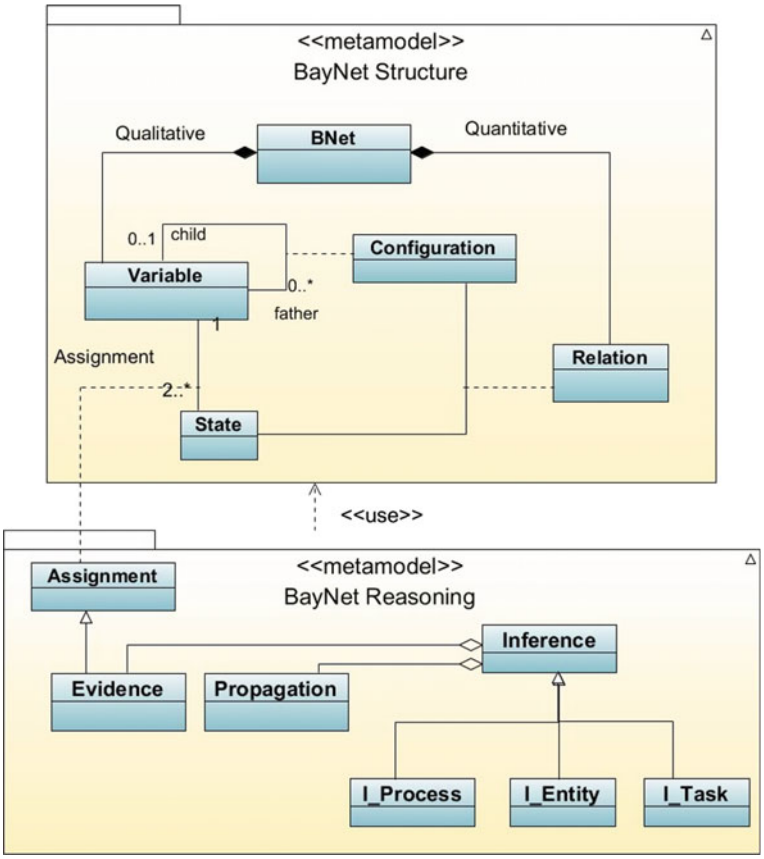


Fig. 6 BayNet metamodel

The *BayNet* metamodel is the basis that will provide a specific and intuitive notation for modeling BN-based KBS and including it in a UML project. UML profiles are UML packages with the stereotype <<*profile*>>. A profile can extend a metamodel or another profile while preserving the syntax and semantic of existing UML elements. The aim of UML Bayesian network profile (*UBN*) is to define a language for designing, visualizing, specifying, analyzing, constructing and documenting a BN KBS.

The *UBN* is defined mapping the *BayNet* metamodel to UML metaclasses, making the necessary extensions at semantic level. Most of concepts are mapped to stereotypes on a selected UML metaclass, such as class, use case, association or operation. Also we have defined icons for the stereotypes. Icons allow modelers to use intuitive symbols instead of UML shapes. The package *UBN* in Fig. 7 shows the actual mapping with UML metaclasses.

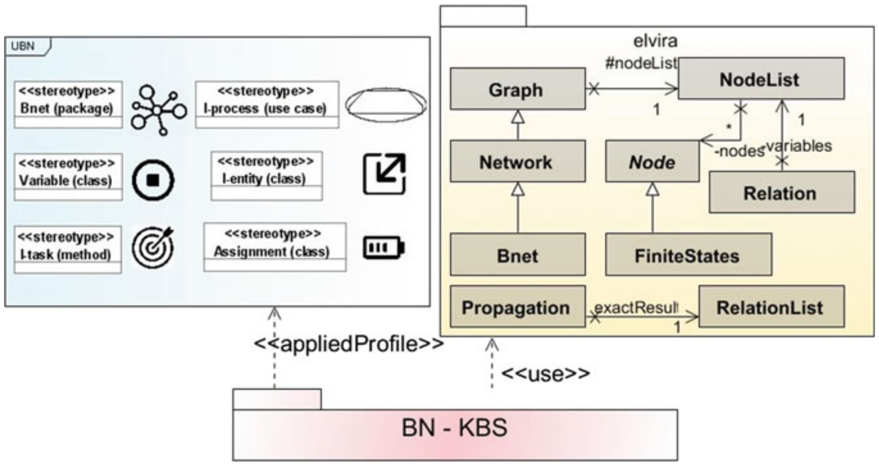

UBN
<<stereotype>> Bnet (package)
<<stereotype>> I-process (use case)
<<stereotype>> Variable (class)
<<stereotype>> I-entity (class)
<<stereotype>> I-task (method)
<<stereotype>> Assignment (class)
elvira
Graph
#nodeList
NodeList
1
*
1
-nodes
variables
Network
Node
Relation
Bnet
FiniteStates
Propagation
exactResult
RelationList
1
<<appliedProfile>>
<<use>>
BN - KBS

Fig. 7 Modeling packages for Bayesian network KBS development

Once the profile is defined, it can be used in the software development of a particular application by defining a stereotyped dependency (<<*applyProfile*>>) between the *UBN* package an the package that is being under development for the application, as Fig. 7 shows. A partial view of the class model of Elvira is included as it is needed in order to define the M2M translation between PIM and PSM (see Fig. 4).

5.3 BNs KBS Process Model Using UBN

The three main steps in the development of software systems that embed functionalities based and not based on knowledge, concerning the decision support process and the information management processes, are: *Requirement modeling* (RM), *Expert modeling* (EM) and *Specification of the software solution* (SSS) [11]. The first two are in charge of the definition on the PIM model according MDE (see Fig. 4), and here is where *UBN* gets its value, because we can use only UML in order to execute RM and EM, an unified language.

The first activity consists of collecting, structuring and organizing all the relevant data for a concise and accurate definition of the problem to be solved by the software system covering user's needs, either based or non-based on knowledge. User requirements are modeled with UML use cases whereas data managed by the application are modeled using class diagrams. Some of the modeling elements are stereotyped using *UBN* if they involve some kind of knowledge. For instance, some classes of the class diagram are stereotyped as candidates variables in a BN (i.e. <<*Variable*>> stereotype), which correspond variable identification step in the process to build a

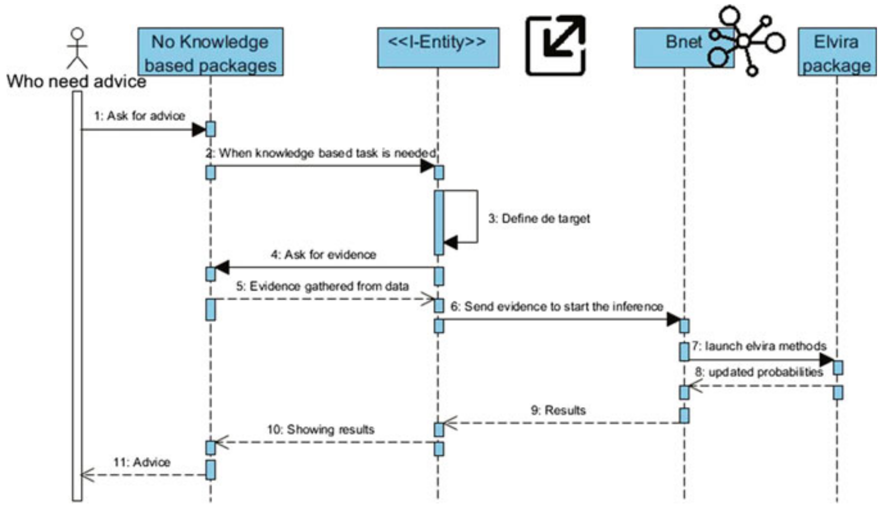


Fig. 8 Model for a BN-inference execution using stereotypes

BN. Relationships between variables (i.e. structure elicitation step) are included in the PIM as stereotyped associations of type <<*father*>>. Once the BN structure has been established, the probabilities are estimated (i.e. parameter elicitation step) creating instances of *Configuration*.

Each inference process has to be modeled as an $I\_Process$, which is an extension of use cases, that is, it represents an interaction that uses a BN (i.e. an <<*BNet*>>). This kind of use case has also to manage a class that encapsulates all the methods relate to the inference, $<< I\_Entity >>$. When developers are building the PIM model of the software application, they must stereotype some of the identified use cases as $<< I\_Process >>$ in case these interactions were based on knowledge. Besides, several new classes have to be defined in order to control the inference tasks. Figure 8 shows, using a informal sequence diagram, the pattern of the messages chain that have to be followed to perform a reasoning task using a $BNet$.

Once the BN and the reasoning processes have been modeled, the PIM model is ready to be translated to code. At the end of MDE chain, the specification of the software solution (SSS) represents a M2M translation that produces the PSM. Based on the PSM obtained, a M2C translation can be used to obtain a BN-based KBS (see Fig. 4).

6 Case Study: A Pest Control BN-Based KBS

This section shows how to apply *UBN* in an specific KBS development project by applying an architectural model for agrarian software [16]. Let us begin with a brief description of the project to assist decision making in an agricultural domain. Our

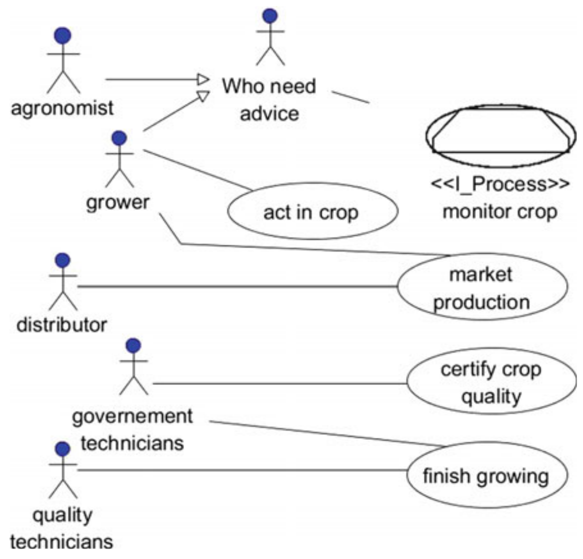


Fig. 9 Use cases for the pest control case study

problem is related to pest control in a given crop under the regulation of Integrated Production. The Integrated Production Quality standard is adopted by a group of growers in order to achieve a quality production certification. The adoption of this standard forces growers to be disciplined in growing which involves intervention by technicians, marketing controls, and periodical inspection by the standard certification agencies.

The typical processes in an integrated production problem are shown in Fig. 9. All tasks related to information required for quality management standards, without need an knowledge based approach, are: Market Production, Act in Crop, Certify Crop Quality, and Finish Growing. All tasks related to pest control are performed by growers and agronomists inside the Monitor crop process and represents the inference tasks. That is, Monitor crop needs to manage expert knowledge that can be modeled using a KE representation, a previous work models it using rules [5, 14], but we propose to model it by means of a BN. This expert modeling approach has been successfully applied to determine the need of applying a treatment for the olives' fly (dacus olae) [38].

The decision process when monitoring a crop is made at two levels. First, a decision is made on whether crop control action is necessary by sampling pests and estimating risk of attack. Then, if it is decided that crop control action is required, the product (chemical or biological) to be applied has to be selected. The treatment advised has to respect natural enemies and other biological products previously used.

Monitor crop use case can be described as the following informal scenario: Each week, the grower or an agronomist samples the crop conditions and makes an estimate of the risk of pest attack. Crop sampling consists of direct observations and count of harmful agents in randomly selected plants. Where an imbalance is detected, the

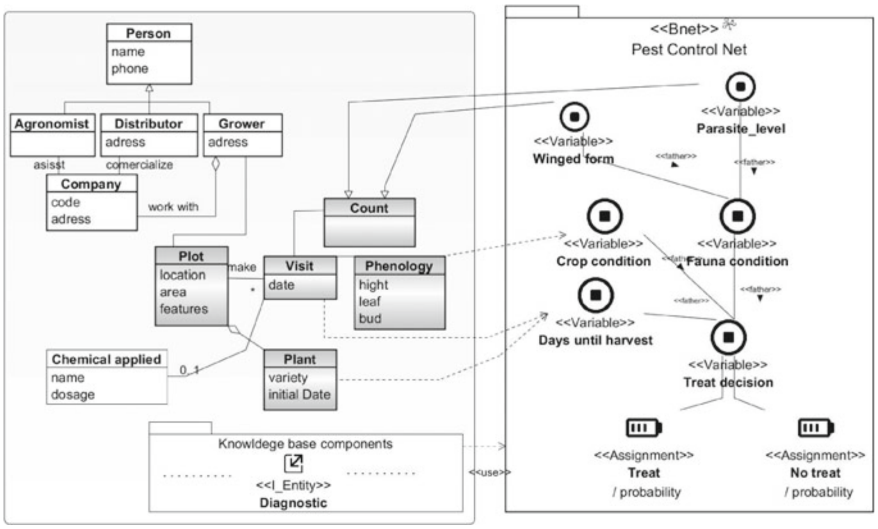


Fig. 10 Partial view of the PIM for the BN-based KBS

expert advises a treatment that meets the integrated production standard. It is the only integrated production process that involve the use of knowledge, so it is stereotyped as an $<< I\_Process >>$.

A first version of the PIM is shown in Fig. 10. A crop is a complex system consisting of a plot of land, plants, a set of diseases and pests, and natural enemies that may be able to control them. The problem is to decide what treatment to apply, in order to maintain a balanced system. Figure 10, shows the UML class diagram obtained. Some of the classes in the model are variables of the Bayesian network (EM). When an agronomist visits a greenhouse, he writes down the date of the visit and samples the crop, including information about the crop phenology, fauna, weather (wind, rain, etc.), environment (weeds) and the number of harmful agents in plants.

Some of the classes are stereotyped as <<*Variable*>>. Some of them have direct connection with attributes of a class, for instance crop-condition is measured in terms of its phenology. The crop-condition, along with the intensity of pest attack (represented by the variable fauna-condition) determines the need for applying a plant health treatment or not. This last variable is related to the number of winged-forms of the pathogen that increases the infestation risk, and also the number of parasited pathogens that reduces the infestation problem. Both variables are extracted from class Count. The periodic sampling inspections are performed weekly.

At this point we have identified the structure of the BN and what are the processes that use the network (i.e. monitor crop). The BN construction using *UBN* concludes with the estimation of probabilities from local government database of cases creating the appropriate instances of *Configuration*.

7 Conclusions

In this work we present a proposal that allows to manage a domain-specific language for BN without conflicts with UML semantics. This can be viewed as a general framework to apply MDE, extending it to the BN-based KBS development. We have presented a metamodel (*BayNet*) and an UML profile (*UBN*) for BN-based KBS modeling. Developing a profile is a difficult task that implies to perform many steps, so that we have split the metamodel in packages in order to facilitate the evolution of the proposal. This metamodel covers several important aspects for achieving the seamless inclusion of BN models into a final software solution for an organizational environment. We have included a case study showing the applicability of our MDE chain, which correspond to a simplified version of a real world problem: integrated production in agriculture.

The next steps of this research will focus on validating the profile using a CASE tool, integrating this MDE solution with the rules of MDE solutions approaches previously done, and also testing it in a real-life development project that includes knowledge-based features.

Acknowledgements This research has been financed by the Spanish Ministry of Economy and Competitiveness under project TIN2013-46638-C3-1-P and partially supported by Data, Knowledge and Software Engineering (DKSE) research group (TIC-181) of the University of Almería.

References

1. Abdullah, M., Benest, I., Paige, R., Kimble, C.: Using Unified Modeling Language for Conceptual Modelling of Knowledge-Based Systems. In: Parent, C., Schewe, K.-D., Storey, V., Thalheim, B. (eds.) Conceptual Modeling - ER 2007. Lecture Notes in Computer Science, vol. 4801, pp. 438–453. Springer, Heidelberg (2007)
2. Berners-Lee, T., Hendler, J., Lassila, O.: The semantic web. Sci. Am. **284**(5), 28–37 (2001)
3. Buchanan, B.G., Barstow, D., Bechtal, R., Bennett, J., Clancey, W., Kulikowski, C., Mitchell, T., Waterman, D.A.: Constructing an expert system. Build. Expert Syst. **50**, 127–167 (1983)
4. Buntine, W.L.: A guide to the literature on learning probabilistic networks from data. Knowledge and data engineering. IEEE Trans. Knowl. Data Eng. **8**(2), 195–210 (1996)
5. Cañadas, J., del Águila, I.M., Palma, J.: Development of a web tool for action threshold evaluation in table grape pest management. Precis. Agric. 1–23 (to appear) (2016)
6. Cañadas, J., Palma, J., Túnez, S.: A tool for MDD of rule-based web applications based on OWL and SWRL. In: Nalepa, G.J., Baumeister, J. (eds.) Proceedings of the 6th Workshop on Knowledge Engineering and Software Engineering, vol. 636. http://CEUR-WS.org (2010)
7. Cañadas, J., Palma, J., Túnez, S.: Defining the semantics of rule-based Web applications through model-driven development. Appl. Math. Comput. Sci. **21**(1), 41–55 (2011)
8. Cestnik, B.: Estimating probabilities: a crucial task in machine learning. In: Proceedings of the European Conference on Artificial Inteligence (ECAI'90), pp. 147–149 (1990)
9. Cooper, G.F., Herskovits, E.: A bayesian method for the induction of probabilistic networks from data. Mach. Learn. **9**, 309–347 (1992)

10. del Águila, I.M., Cañadas, J., Palma, J., Túnez, S.: Towards a methodology for hybrid systems software development. In: Proceedings of the Eighteenth International Conference on Software Engineering & Knowledge Engineering (SEKE'2006), pp. 188–193. San Francisco (2006)
11. del Águila, I.M., del Sagrado, J., Túnez, S., Orellana, F.J.: Seamless software development for systems based on bayesian networks - an agricultural pest control system example. In: Moinhos-Cordeiro, J.A., Virvou, M., Shishkov, B. (eds.) ICSOFT 2010 Proceedings of the Fifth International Conference on Software and Data Technologies, Vol. 2, pp. 456–461. SciTePress (2010)
12. del Águila, I.M., del Sagrado, J.: Metamodeling of Bayesian networks for decision-support systems development. In: Nalepa, G.J., Cañadas, J. Baumeister, J. (eds.) KESE 2012 Proceedings of 8th Workshop on Knowledge Engineering and Software Engineering at the 20th Biennial European Conference on Artificial Intelligence (ECAI 2012), CEUR Workshop proceedings, vol. 949, pp. 12–19 (2010)
13. del Águila, I.M., Palma, J., Túnez, S.: Milestones in software engineering and knowledge engineering history: a comparative review. Sci. World J. (2014)
14. del Águila, I.M., Cañadas, J., Túnez, S.: Decision making models embedded into a web-based tool for assessing pest infestation risk. Biosyst. Eng. **133**, 102–115 (2015)
15. del Sagrado, J., del Águila, I.M., Orellana, F.J.: Architecture for the use of synergies between knowledge engineering and requirements engineering. In: Lozano, J.A., Gámez, J.A., Moreno, J.A. (eds.) Advances in Artificial Intelligence, vol. 7023, pp. 213–222. Springer, Heidelberg (2011)
16. del Sagrado, J., Túnez, S., del Águila, I.M., Orellana, F.J.: Architectural model for agrarian software management with decision support features. Adv. Sci. Lett. **19**(10), 2958–2961 (2013)
17. Drapeau, S., Madiot, F., Brazeau, J.F., Dugré, P.L.: SmartEA: Una herramienta de arquitectura empresarial basada en las técnicas MDE. Novática **228**, 21–28 (2014)
18. Druzdzel, M.J., Roger, R.F.: Decision support systems. In: Broy, A.K. (ed.) Encyclopedia of Library and Information Science, pp. 120–133. Marcel Dekker, New York (2000)
19. Elvira Consortium: Elvira: an environment for creating and using probabilistic graphical models. In: Gámez, J.A., Salmerón, A. (eds.) Proceedings of the First European Workshop on Probabilistic Graphical Models (PGM-02), pp. 223–230 (2002)
20. Ernst, G.W., Newell, A.: GPS: A Case Study in Generality and Problem Solving. Academic Press, Cambridge (1969)
21. Fikes, R.E., Nilsson, N.J.: STRIPS: a new approach to the application of theorem proving to problem solving. Artif. Intell. **2**(3–4), 189–208 (1971)
22. Gašević, D., Djurić, D., Devedžić, V.: Model Driven Architecture and Ontology Development. Springer, New York (2006)
23. Giachetti, G., Valverde, F., Pastor, O.: Improving automatic UML2 profile generation for MDA industrial development. In: Song, I.Y., et al. (eds.) Advances in Conceptual Modeling - Challenges and Opportunities, ER 2008 Workshops. Lecture Notes in Computer Science, vol. 5232, pp. 113–122. Springer, Heidelberg (2008)
24. Gómez-Pérez, A., Fernández-López, M., Corcho, O.: Ontological Engineering: With Examples From the Areas of Knowledge Management. E-Commerce and the Semantic Web. Springer, Heidelberg (2006)
25. Harman, M., Mansouri, S.A., Zhang, Y.: Search-based software engineering: trends, techniques and applications. ACM Comput. Surv. **45**(1), A1–64 (2012)
26. Hart, P.E., Duda, R.O., Einaudi, M.T.: PROSPECTOR a computer-based consultation system for mineral exploration. Math. Geol. **10**(5), 589–610 (1978)
27. Hussmann, H., Meixner, G., Zuehlke, D.: Model-Driven Development of Advanced User Interfaces. Springer, New York (2011)
28. Jensen, F.V., Nielsen, T.D.: Bayesian Networks and Decision Graphs. Springer, New York (2007)
29. Kaufman, L., Rousseeuw, P.J.: Finding Groups in Data: An Introduction to Cluster Analysis. Wiley, New York (2005)

30. Kjaerulff, U.B., Madsen, A.L.: Bayesian Networks and Influence Diagrams: A Guide to Construction and Analysis. Springer, New York (2008)
31. Koller, D., Friedman, N.: Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning. The MIT Press, Cambridge (2009)
32. Korb, K.B., Nicholson, A.E.: Bayesian Artificial Intelligence. CRC Press, Boca Raton (2010)
33. Maher, M.L., Allen, R.H.: Expert System Components. In: Maher, M.L. (ed.) Expert Systems for Civil Engineers: Technology and Application, pp. 3–13. American Society of Civil Engineering (1987)
34. Moreno, N., Romero, J.R., Vallecillo, A.: An overview of model-driven web engineering and the MDA. In: Rossi, G., Pastor, O., Schwabe, D., Olsina, L. (eds.) Web Engineering: Modelling and Implementing Web Applications, pp. 353–382. Springer, London (2008)
35. Neapolitan, R.E.: Learning Bayesian Networks. Prentice-Hall, New Jersey (2004)
36. Newell, A.: The knowledge level. Artif. Intell. **18**(1), 87–127 (1982)
37. Object Management Group.: MDA Guide Version 1.0.1. OMG document: omg/2003-06-01 (2003)
38. Orellana, F.J., del Sagrado, J., del Águila, I.M.: SAIFA: a web-based system for integrated production of olive cultivation. Comput. Electron. Agric. **78**(2), 231–237 (2011)
39. Papajorgji, P.J., Pardalos, P.M.: Software Engineering Techniques Applied to Agricultural Systems: An Object-Oriented and UML Approach. Springer, New York (2014)
40. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann Publishers Inc., San Francisco (1988)
41. Schmidt, D.C.: Guest editor's introduction: model-driven engineering. Computer **39**(2), 25–31 (2006)
42. Shortliffe, E.H.: MYCIN: Computer-Based Medical Consultations. Elsevier, New York (1976)
43. Spirtes, P., Glymour, C., Scheines, R.: An algorithm for fast recovery of sparse causal graphs. Soc. Sci. Comput. Rev. **9**, 62–72 (1991)
44. Studer, R., Benjamins, V.R., Fensel, D.: Knowledge engineering: principles and methods. Data Knowl. Eng. **25**(1), 161–197 (1998)
45. Studer, R., Fensel, D., Decker, S., Benjamins, V.R.: Knowledge engineering: survey and future directions. In: German Conference on Knowledge-Based Systems, pp. 1–23. Springer (1999)

Knowledge Acquisition During Software Development: Modeling with Anti-patterns

Paraskevi Smiari, Stamatia Bibi and Ioannis Stamelos

Abstract Knowledge is a strategic resource; that should be timely acquired and exploited to manage and control software development. Software development is a knowledge intensive process characterized by increased uncertainty, presenting large variations among different development environments. Project uncertainty and volatility confounds the traditional knowledge-based processes since at any time traditional software project management techniques and patterns may be considered out of scope. In this chapter a dynamic and constantly adaptive knowledge encapsulation framework is presented. This framework analytically describes (a) metric collection methods along with metrics that attribute to knowledge creation regarding successful software development (b) representation mechanisms of the knowledge created in the form of anti-patterns (c) Bayesian Network analysis technique for converting the data to knowledge allowing inference mechanisms for testing the applicability of the anti-pattern. The presented approach is demonstrated into a case study showing both its feasibility and applicability.

1 Introduction

Software development is a cross-disciplinary cognitive activity requiring knowledge from several different domains [23]. Human knowledge-creating activities and past experience should contribute to the constantly evolving software development

P. Smiari (✉) · S. Bibi
Department of Informatics & Telecommunications Engineering, University of Western Macedonia, Kozani, Greece
e-mail: psmiari@uowm.gr

S. Bibi
e-mail: sbibi@uowm.gr

I. Stamelos
Department of Computer Science, Aristotle University of Thessaloniki, Thessaloniki, Greece
e-mail: stamelos@csd.auth.gr

G.J. Nalepa and J. Baumeister (eds.), *Synergies Between Knowledge Engineering and Software Engineering*, Advances in Intelligent Systems and Computing 626, https://doi.org/10.1007/978-3-319-64161-4_4

knowledge base according to Davenport et al. [8]. The key to systematic software development and process improvement is the decent management of knowledge and experience. Many techniques [16, 19] have been applied to manage knowledge regarding software development in order to improve aspects of the whole process such as, cost estimation and project management, fault identification and quality of the outcome. These aspects are compound by vulnerabilities such as changing requirements, team dynamics and high staff turnover [1].

Capturing and representing software project management knowledge while catering for its evaluation are some of the mechanisms of great importance. This book chapter provides a framework for effectively capturing the knowledge created during software development. This framework consists of three distinct phases:

- The first phase includes the data collection process that could attribute to knowledge creation. Adopting the appropriate qualitative and quantitative data collection approaches is of major importance for capturing the relevant information.
- The second phase is related to transforming the information and data collected from the previous step in the form of anti-patterns. Each anti-pattern represents a management problem and will be described as a set of symptoms, suggested solutions and identification factors.
- The final phase includes the representation of the anti-patterns created in the form of a Bayesian Network (BN) in order to be able to proceed to the validation process of the derived knowledge.

The final output of this framework will be an anti-pattern knowledge base in the form of Bayesian Networks available to managers and developers that can serve as a management toolkit.

Software project management anti-patterns propose frequently occurring solutions [5] to problems that have to do with flawed behavior of managers or extensive management practices that prevent a software project from successful [15] implementation. Nonetheless, the documentation of anti-patterns is conducted using informal or semi-formal [9] structures that do not readily encourage the reusability and sharing of knowledge. In addition, the number of defined anti-patterns and the number of printed documentation is expanding to the point that it becomes difficult for it to be effectively used. Thus better structured anti-pattern representations are required in order for them to become a widespread practice.

By applying the BN [18] representation formalism to anti-patterns we can gain useful insight about the knowledge created during software development and also perform post mortem analysis. The BN model offers a convenient mechanism to model and disseminate knowledge regarding software management anti-patterns which incorporates uncertainty. Re-factorings proposed by the anti-patterns can be tested to view their reflection to the process. The suggested model can be used by project managers to illustrate the effect of an anti-pattern solution in the process.

In order to reinforce the applicability of the proposed framework a case study is presented that exemplifies the process of creating the anti-pattern BN models. In particular, we use as a pilot the “BENEFIT” project, an ambitious, highly innovative cross disciplinary crowd-sourcing platform for tourism marketing, where the "tech-aware" anti-pattern was actually formulated and assessed. In brief the problem that BENEFIT project was faced from the early stages of implementation was that there were many technical conflicts and disagreements that inhibited project progress. Four new metrics were introduced that described better the development status namely: team synthesis, organizational structure, project integration and product innovation. The "tech-aware" anti-pattern was then formulated to test the impact of changing the organizational structure in order to better monitor technical conflicts.

This paper is organized as follows: Sect. 2 describes analytically the three steps of the knowledge based framework providing guidelines for acquiring and modeling the knowledge in the form of BN anti-pattern models. Section 3 presents the BENEFIT case study were the three steps of knowledge acquisition and assessment are exemplified. Finally, in Sect. 4, we conclude the chapter and summarize the knowledge acquisition framework.

2 The Knowledge Acquisition Framework

In this section, we present the framework that is suggested for guiding the process of acquiring knowledge during software development. The framework consists of three distinct phases (a) collecting data during development (b) modeling the data in the form of anti-patterns (c) representing the anti-pattern with the appropriate BN model and assessing its applicability. Therefore, in Sect. 2.1 possible methods for collecting data along with the relevant metrics that can be valuable in assessing a software process are presented, in Sect. 2.2 the representation formalism of anti-patterns is described and in Sect. 2.3 the Bayesian Networks method is presented.

2.1 Collecting the Relevant Data

In this section we describe the methods that can be used to acquire the relevant data that can help us realistically capture the software development status quo of a team. Additionally possible sources of data and metrics are recorded in order to offer practitioners a library of metrics that can help in acquiring knowledge relevant to software development processes.

2.1.1 Data Collection Methods

In this section two complementary types of data collection methods will be presented namely qualitative and quantitative methods [10]. Both of these methods can be used to record primary data, data coming directly from the source (respondents, activities logged, crowd sourcing, etc.) or secondary data, historical data that come from experiments, aggregations and measurements.

Data collection approaches for qualitative research usually involve direct interaction and contact of the observer with the key individuals or groups whose opinion is considered valuable. Among the qualitative methods we can find (a) interviews (b) participant observation (c) focus groups (d) questionnaires/testing.

Interviews, which can be structured, semi-structured or unstructured are used to collect experience, opinion and emotions of the interviewee. A software project manager, formally or informally, "interviews" the members of the development team and holds the behavioral aspects of the responses. It should be mentioned that from the interviews large amount of data can be derived that can provide the "beat" of the development team. These data are rich, provide a wealth of information coming from the primary actors of the development process but due to their intangible nature remain often unrecorded and therefore unexploited. **Focus groups** method actually is a form of observation of a group of people that can interact and exchange opinions letting us gain useful insight regarding the participants' behavior and attitude. On the other hand modern paradigms of this method include data coming from online social networking platforms such as company blogs, employees' facebook, collaborative development tools, etc. **Participant observation** involves the intensive involvement of the observer with the software development team. Finally **questionnaires**, is among the most common methods to collect data that records the opinion of the respondents in the form of answers that can be unstructured or calibrated in a certain scale. In the case of software development process we should mention that interviews in the form of discussions, focus groups as blogs and collaborative tools and short questionnaire completion during and after the development of a project could provide a valuable source of data.

Quantitative methods on the other hand are less personalized and more unbiased. Among quantitative methods we can find (a) surveys/polls (b) automatic collection of data. **Surveys** are like questionnaires but they are completed anonymously, usually without personal contact and usually involve a large sample. In the case of project development teams that would mean that a survey could be an anonymous evaluation of certain aspects of the process during or after the completion of the project. **Automatic methods** may include direct data coming from systems that monitor or aid the development process and record information regarding the development progress (log files, nof active members, number of teleconference held etc.). Quantitative methods are more accurate and present greater objectivity compared to qualitative methods. Still both types of data collection methods are required since both human perceptions and numerical descriptions are necessary for fully recording the status of a software development process.

2.1.2 Software Development Metrics

Since software development process can present significant deviations compared to the traditional models found in literature [7] being able to identify the relevant metrics that will help us monitor, assess, update and finally improve a process is very crucial.

Table 1 People drivers

People drivers	Driver	Metric
Experience	Analysts capabilities	1–5 Scale
	Programmers experience	1–5 Scale
	Familiarity with the problem domain	1–5 Scale
Cultural issues	Reward mechanisms	1–5 Scale
	Collaboration	1–5 Scale
	Capable leadership	1–5 Scale
Stakeholders participation	Customer participation	1–5 Scale
	End-user involvement	1–5 Scale

The classical 4 Ps in software project management, **people, product, process** and **project** [10] can help us identify the relevant metrics that can better describe the software process.

Metrics regarding the stakeholders of a software development project are necessary to address the coordination and cooperation efficiency of the team. Among the groups of **people** that affect the team efficiency are the senior manager, the project manager, the development team with all discrete roles (analyst, programmer, tester), the customer and the end-users. Metrics that can describe the efficacy of each role are presented in Table 1. Typical examples of metrics describing the people involved in software development are those expressing the experience of the team (analysts' capabilities, familiarity with the application domain). Cultural characteristics also affect the performance of a team as well organized teams whose leadership encourage communication and knowledge exchange offering rewards are more productive compared to impersonal teams. User and customer involvement is also appointed by recent studies as a critical success factor of software projects. Therefore, customer participation and end user involvement are among the parameters that need to be quantified.

Measurement of **process** attributes is important for establishing a baseline for assessing improvements and identifying possible process flaws. All methods, techniques, tools and supplements that may be used to support the development process should be quantified and recorded in order to be able to measure the efficiency of the development process. Among these attributes the use of CASE (Computer Aided Software Engineering) tools, the utilization of models, techniques and standards are the main aspects that define the level of support and observation of the development procedure. Following a well-defined and guided process customized to each company needs is crucial for delivering quality software within time and budget constraints. To achieve this target the process should be constantly measured and improved to reach the quality goals of the company. Table 2 presents process drivers.

Project attributes related to a software project include relevant variables that describe the type and the size of the project. The aggregation of project variables offers an indicator of the complexity of the project preparing the management for the risks and difficulties that may appear. Project attributes can be descriptive variables

Table 2 Process drivers

Process drivers	Driver	Metric
Use of case tools	Versioning tools	% of usage
	Analysis & design tools	% of usage
	Testing tools	% of usage
Management process	Use of lifecycle models	Yes or No
	Managed development schedule	1–5 Scale
Methodologies	Existence of best practices	1–5 Scale
	Software reuse	% of the total LOC

Table 3 Project drivers

Project drivers	Driver	Metric
Type of project	Application type	ERP, Telecom, Logistics, etc.
	Business type	Medical, Public Sector, Transports, Media, etc.
	Development type	New development, Re-development, Enhance
User type	Level of usage	Amateur, Professional, Casual
	Number of users	1–50, 50–200, 200–1000, >1000
Size	Source code lines	Lines of code (LOC)
	Function points	Number of function points

Table 4 Product drivers

Product drivers	Drivers	Metric
Technical attributes	Distributed databases	1–5 Scale
	On-line processing	1–5 Scale
	Data communications	1–5 Scale
	Back-ups	1–5 Scale
	Memory constraints	1–5 Scale
	Use of new, immature technologies	1–5 Scale
Non-functional requirements	Reliability	1–5 Scale
	Performance	1–5 Scale
	Installation ease	1–5 Scale
	Usability	1–5 Scale
	Security	1–5 Scale

referring to the development type of the project, the application type and the user type of the application. Size attributes can be an initial assessment of functional requirements measured in function points or at later stages in the development process measured as Lines of Code [4]. Table 3 summarizes project drivers.

Product attributes are constraints imposed on the software by the hardware platform and the utilization environment. Such constraints include run-time performance, memory utilization, performance standards and transaction rates. Table 4 summarizes product metrics.

2.2 Forming the Anti-pattern

A project manager needs to ensure that the management of the 4 Ps as described earlier is carried out effectively without any arising issues. In the occurrence of these issues, anti-patterns play a significant role due to their ability to describe commonly occurring solutions to the problems that lead to undesired results [5, 21]. Anti-patterns propose re-factored solutions that can combat problems with reference to flawed behavior of managers or pervasive management practices that constrain a software project from being successful [15]. Any reader who is not acquainted with anti-patterns can start with [5, 15] as introductions to the matter.

One way, to identify potential problems and provide a refactored solution in a practical and reusable manner, is by capturing and representing tacit knowledge in the form of an anti-pattern template. An anti-pattern template [15] is an informal presentation of the anti-pattern that depicts the management dysfunction and the remedies for all those engaged. The anti-pattern template carries out insight into the causes, symptoms, consequences and identification of the problem suggesting band-aids and refactoring. For example in Table 5 the encoding of the problems related with choosing the optimal agile team size is done by the “Is Five the Optimal Team Size?” anti-pattern [12] (Table 6). The anti-pattern is described by a central concept, the dysfunction that it presents, a short explanation of the causes of the problem, a band-aid that suggests a short-term solution strategy, self-repair mechanisms for selecting the appropriate solutions, re factoring solutions and finally identification questions that will help a manager verify whether his team is suffering from the particular anti-pattern.

Table 5 Pattern “Is Five the Optimal Team Size” central concept

Name	“Is Five the Optimal Team Size?”
Central concept	It is agreed by most Agilists that smaller teams can be more functional and productive in comparison to larger teams. The definition of the optimal team size is, however, still a challenge. In order to produce more code, large teams are still being used. Nonetheless, a team size of 5 [12] seems to satisfy all the conditions related to Scrum recommendations, Parkinson’s Law, natural limit of short term memory and favorable communication channels. Software managers neglect to comprehend the influence of organizational and environmental matters on choosing the ideal team size
Dysfunction	This anti-pattern is available to the board of managers or the software project manager, who picked the amount of an agile team without taking into account the characteristics of the organization and/or the software project
Explanation	The purpose of this anti-pattern is the decision that the optimal team size is a team of 5 or a large team in order to produce more code

Table 6 "Is Five the Optimal Team Size" anti-pattern refactoring

Band aid	No band aid exists for this anti-pattern. No short term management strategy can be used to handle this problem
Self repair	Project managers need to consider the environmental and organizational concerns that influence the ideal team size. They should start by identifying key issues that influence the success of a software project given the chosen team size
Refactoring	It is important that before determining the size of an agile team. Software project managers should first determine the concerns that influence the optimal team size and how these concerns affect the progress of the project. Software managers have to accept that there is no generic team size that can be used for all agile projects and that each project has exclusive characteristics that need to be taken into account. Management needs to look out for the team location (collocated or not), the organization size, iteration length, and the existence of offshoring teams. Past project data can be used to detect how different team sizes behave in the same company. This will grant managers help to calculate the impact of the chosen team size on the success of the project
Identification	The following questions should be answered with a "Yes" or "No" • Has the organization used the same agile team size repeatedly? • Does the agile project manager always use a team size of 5? • Does the agile project manager always use a large team size? If you responded "Yes" to one or more of these statements, your organization is probably suffering from "Is Five the Optimal Team Size?"

2.3 *Modeling the Anti-patterns with Bayesian Networks*

In this section we present the Bayesian Network Models background theory [13] and provide a short example coming from the project management domain. Bayesian Network models known also as Bayesian Belief Networks are casual networks that form a graphical structure. These graphical structures consist of nodes and links between those nodes without, however, forming a cycle with each other. This is the reason why they belong in the Directed Acyclic Graphs (DAGs) family and these graphical structures are popular in the fields of statistics, machine learning, artificial intelligence and are used to handle uncertainty in software development application domains as process modeling [2, 3, 11], defect prediction [17] and cost estimation [14, 22]. Specifically, each node represents a random variable that has a finite set of mutually exclusive states. Furthermore, each link represents probabilistic cause-effect relations between the linked variables.

This type of network is used to track how a change of certainty in one variable can cause an effect on the certainty of other variables. The relation that links the two nodes can be seen on Bayes' rule:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

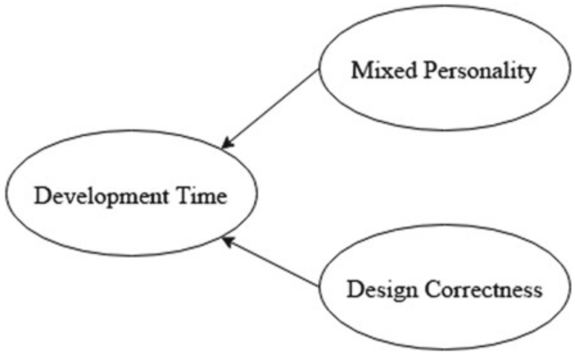


Fig. 1 A BN example for modeling "Development Time"

For each node A with parents B1, B2,…,Bn there is attached an NxM Node Probability Table (NPT), where N is the number of node states and M is the product of its cause-nodes states. In this table, each column represents a conditional probability distribution and its values sum up to 1.

Example

In Fig. 1 we present a simple BN model were two nodes "Mixed Personality" and "Design Correctness" affect the node "Development Time". Mixed Personality is an indicator of heterogeneous developer personality and its states are Yes or No. The Design Correctness measures the points in 1–10 scale, obtained by each pair for all the tasks and is divided in two categories, False for points below 5.4 and True for points above 5.4. The Development Time indicates the total time that took the pair to complete the tasks and it is divided in two categories, Low for total time below 90 min and High for total time above 90 min.

The BN model contains 3 variables and 2 connecting links. We have one converging variable connection. In a converging connection if nothing is known about Development Time except what may be inferred from knowledge of its influencing variables (parents) Mixed Personality and Design Correctness, then the parents are independent. This means that evidence on one of them has no influence on the certainty of the others. If Development Time changes certainty, it allows communication between its parents.

Specifically, if Mixed Personality is Yes and Design Correctness is True, then there is a 20% probability that the Development Time will be Low and 80% probability that it will be High (Table 7).

3 The Case Study

This section exemplifies the framework of Sect. 2 by presenting an actual case where the knowledge acquired during software development were modeled in the form of anti-pattern to be diffused and assimilated into new projects. In this section we will

Table 7 Node probability table for the BN model of Fig. 1.

	Mixed personality	Yes		No	
	Design correctness	True	False	True	False
Development time	Low	0.2	0.7	0.8	0.4
	High	0.8	0.3	0.2	0.6

describe analytically the software development project under study. In Sect. 3.2 we will present the data that were collected during development, in Sect. 3.3 the new anti-pattern model and in Sect. 3.4 the knowledge-based models representing the proposed anti-pattern.

3.1 *CASE STUDY: The BENEFIT Platform*

The BENEFIT Platform was designed to provide a solution catering to arise brand marketing, awareness and advertising capability of the tourism sector by providing an on-line company platform that will offer (a) specialized marketing toolkits available to the wider public, (b) advanced crowdsourcing tools to process tourism experience and review data in order to extract and present collective knowledge, (c) advanced forecasting models exploiting the tourism market sentiment to identify market trends and threats, (d) novel personalized recommendation systems to support marketing decisions according to the company's profile.

The project was funded by a Greek local tourism association in order to promote the tourism marketing of the area and arise awareness of the local tourism business sector regarding the arising trends of crowd- sourced tourism innovations. BENEFIT was an ambitious project and involved the participation of 5 partners with different backgrounds, the first partner (Partner 1) was a university department specializing in the area of Marketing and Business Administration, the second partner (Partner 2) was a university department specializing in Informatics, the third partner (Partner 3) was a Web Company specialized in crowd sourcing applications, the fourth partner (Partner 4) was an on-line marketing company and Partner 5 was the Greek local tourism association. Partner's 1 role was to suggest marketing and socioeconomic models to predict market trends based on the data provided by the crowdsourcing tools, Partners 2 and 3 were responsible for designing and implementing the BENEFIT platform. Partner 4 and Partner 5 would pilot the platform and perform usability tests. All partners were responsible for providing initially the requirements of the BENEFIT platform and defining user stories to be implemented.

For the development of the project three distinct Committees were defined that would manage the development of the BENEFIT platform (a) the Project Coordination Committee (b) the Work packages committees and (c) The Quality Assurance Management Committee. The Project Coordination Committee, consisting of representatives of all partners, was the principal management authority of the BENEFIT project having the final steering and controlling commitment to the project, making decisions on contractual, administrative and technical matters attempting to ensure timeliness and cost effectiveness. The Work Packages Committees was one for each work package and consisted of representatives from partners participating in each work package and were responsible for the monitoring the progress of each WP. The Quality Assurance Management Committee, including representatives from all partners, were responsible for providing internal quality assurance guidance to the BENEFIT project, outlining the policies, the purpose, the organization, the procedures and the responsibilities related to ensuring high quality performance of all activities and also collecting metrics for measuring and improving the process development.

3.2 Data Collection

The Quality Assurance Committee (QAC) due to the fact that the same partners had a long series of projects in which they cooperated was determined to collect data and metrics that would help in the direction of improving the development and implementation process. Therefore from this project and on they decided to create a repository of anti-patterns coming from the "lessons-learned" during each project development [20] summarizing the knowledge created by past projects into the form of anti-patterns and adding also new knowledge created during the development of the BENEFIT implementation.

The QAC team decided that extra metrics would be important to better describe the complexity and specialty of BENEFIT project. Several additional metrics were recorded as the team synthesis, the structural organization, the tools integration complexity and the product innovation. These parameters are analytically described in Table 8 and provided the key to form the anti-pattern presented in Sect. 3.3.

3.3 The "Tech- Aware Manager" Anti-pattern

Due to the technical problems the QAC decided to form a Technical Management Committee who would be responsible for the overall technical management of the project, monitoring the advances performed, ensuring effective coordination of work packages and timely knowledge exchange. The QAC team formed the "tech-aware" antipattern as presented in Tables 9 and 10 to depict the necessity of the Technical Management Committee.

Table 8 The new metrics collected for the BENEFIT project

Metric	Type	Explanation	Possible values
Team synthesis	People	This metric depicts the appropriateness of the team synthesis. Complementary teams consist of members with complementary skills that can work towards a certain direction, heterogeneous teams consist of members that have diverse orientation and homogeneous teams consist of members possessing the skills on the same domain	Complementary teams Heterogeneous teams Homogeneous teams
Structure organization	Process	Represents the organizational structure of the team emphasizing on whether the leader-ship is performed by a management committee, a technical committee or a combination of the two	Management Committee (M) Technical Committee (T) Combination (M+T)
Project integration	Project	The cost of integrating different tools in a single application	Low Average High values (depending on the integration complexity and time required)
Product innovation	Product	Product innovation expresses the difficulty to perform to meet the requirements of an innovative application in our case is the importance of producing accurate estimation results forecasting behaviors on the tourism domain	Low Average High values (depending on the level of innovation required)

The TMC assesses at first-level progress reports, resolve any internal technical conflicts between work packages and plans resources re-allocation if required. The TMC members maintain a constant communication via audio/video conferencing, emails and schedule regular meetings every 3–4 months. The TMC consisted of the Project Coordinator and the Work Package Technical representatives and is chaired by the Technical Manager. The Technical Manager was a senior technical member of the team and was enrolled to lead the technical activities of the project, organize the TMC meetings, prepare the agenda, keep and share the minutes.

Table 9 Pattern "tech- aware" central concept

Name	"tech-aware"
Central concept	The manager of a software development project should possess leadership competencies and abilities to motivate, inspire and encourage the development team, regardless whether he possess technical competencies or not. That is true still that depends on the technical complexity of the project under development. On the other hand is a technical manager enough to lead a complex software application development? May be only in projects with small development teams?
Dysfunction	This anti-pattern is attributable to the project management board that cannot solve technical problems. Symptoms are:
	1. Disagreements regarding technical issues
	2. Conflicts in the allocation of resources
	3. Difficulty in system integration
	4. Limited knowledge dissemination between work packages
Explanation	The cause of this anti pattern is the fact that no technical experts have been formally appointed to monitor development progress and solve technical conflicts

Table 10 The "tech- aware" anti-pattern refactoring

Band-Aid	Form a technical management committee that will contain representatives that possess technical knowledge from all conflicting stakeholders. Appoint a technical manager responsible for taking decisions
Self-Repair	Project managers should examine technical complexity of a project and its size during the early stages of development. If they feel that their technical background is limited within the scope of the project severity they need to form a technical committee
Refactoring	It is essential that before deciding on the organizational structure and management of a project to assess the unique characteristics of the application under development. Software project managers should first identify among others the technical issues that may jeopardize the success of the project, the team competencies, the technological risks, the integration costs. Based on all the special attributes of the project the organizational structure may include a technical committee, a quality assurance team or in more complex projects, an external advisory board. On the other hand in small agile projects the project manager can also be a technician. Software managers need to understand that there is no generic team organizational structure that can be used for all development projects
Identification	The following questions should be answered with a "Yes" or "No" • Does the project depend on immature technology? • Does the project require increased integration efforts? • Is the development team heterogeneous with different back-grounds? • Is the development team disagreeing usually when taking technical decisions? • Is the project relatively complex compared to the other ones that the team has developed? If the answer is yes to at least two of these questions then the development team is suffering from this anti-pattern

After this change in the project organizational structure the development of BENEFIT proceeded without conflicts and internal disagreements as the TMC monitored the development progress and mitigated any technical risks that occurred.

3.4 Knowledge-Based Models of the "Tech-Aware" Anti-pattern

In this section we model the anti-pattern presented in the previous section with the help of Bayesian Networks representation formalism in order to investigate the relationship between the identification factors that help us diagnose the problematic situations and test the impact of the refactored solution on the project progress. The proposed BN model presented in Fig. 2 consists of a set of nodes that represent People, Process, Project and Product drivers (P variables from now and on). Each one of these nodes is affected by the standard metrics described in Tables 1, 2, 3 and 4 whose values are represented cumulatively in the first affecting node (for People we have the node Team Competencies, for Process the node Process Organization, for the Project the node Project Complexity and for the Product the node Product Complexity).

In order to test the influence of the anti-pattern we need to insert in the BN model the new metrics identified in Table 8 as nodes that describe the "tech-aware" anti-pattern and affect the relevant P variables. As a consequence four new nodes are inserted in the model: Team synthesis now affects People node, Organizational

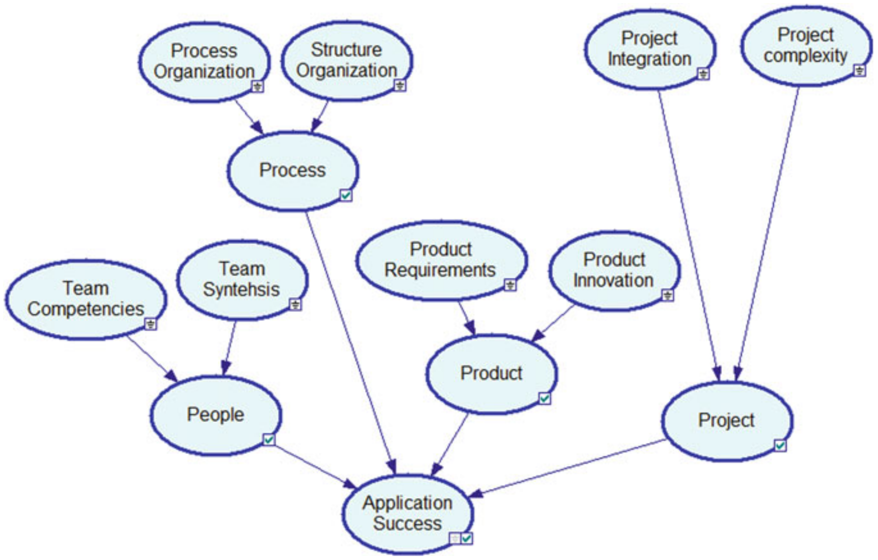


Fig. 2 The BN model of the "tech-aware" anti-pattern

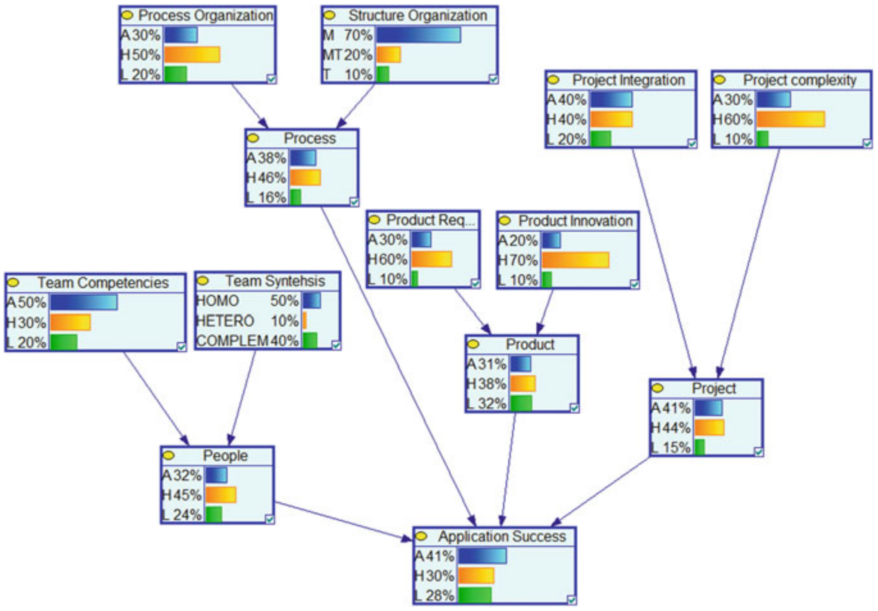


Fig. 3 The BN belief model for the BENEFIT team based on historical data

structure affects Process node, Project Integration affects Project node and Product Innovation affects Product node. At this point the standard drivers that affect application development success are now co-influencing along with the tech-aware anti-pattern nodes the application development success.

Figure 2 represents the Bayesian Network as it is formed after the representations mentioned previously. In this study the values of the Node probability tables were defined based on the historical data collected at phase 1 and on the experience and the knowledge gained during the development of past projects from experts (QAC team) that participated in the BENEFIT team and had in the past a long series of co-operations. The tools used to construct the network and create the node probability tables can be found in [6].

Figure 3 presents the initial belief status of the network based on the node probability tables defined by the experts. For example this network provides the following information regarding the development of projects from the BENEFIT team: In 50% of the projects the team organization was considered to be high while 70% the projects where leaded solely by a manager and in totally the people driver was considered in 46% of the cases to be high meaning that the development team and was well organized and structured. The rest of the information provided by the network is interpreted accordingly.

This Bayes Network can then be useful for applying inference. Certain inferences can be made to show how the change in the values of a metric can affect the values of

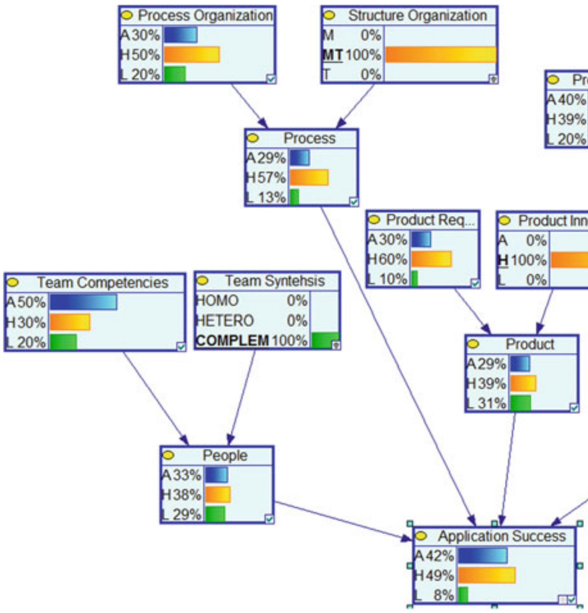


Fig. 4 The BN belief model of the BENEFIT project testing the "tech-aware" anti-pattern

another metric and, finally, reach some conclusions regarding good and bad practices in software project planning that lead to development success.

In order to apply inference, the answers to the identification questions of Tables 9 and 10 are now expressed as values that initialize the anti-pattern nodes to a certain state that can more accurately and objectively describe current status of a project. Figure 3 shows an instance of the Bayes Network when it is instantiated with data coming from the BENEFIT project. We tested the BN model with the actual values of the metrics that represented the BENEFIT project and the Bayesian Network was updated to the one of Fig. 4. The BENEFIT platform was a highly innovative product, incorporating the need of integration of various tools and was developed by a team possessing complementary skills led both by a project management committee and a technical management committee. Changing the values of the relevant nodes we observe that the probabilities of the affected nodes also changed. The refactored solution to the "tech-aware" anti-pattern has a positive impact on the Application Success. On the other hand it would be interesting to test such a change in projects with low innovation, low project complexity and homogeneous development teams. In such cases the probabilities depict that the appointment of technical management committee would be of no value.

4 Conclusions

In this paper we proposed a Knowledgebased framework for acquiring knowledge during software project development and model it in the form of anti-patterns represented by Bayesian Networks. The framework consists of the following phases (a) acquire data during development (b) identify problems and model them in the form of anti-patterns (c) represent and assess the anti-pattern in the form of a Bayesian Network Model. BN modeling is particularly useful and well suited to the domain of anti-patterns because it provides a solid graphical representation of the probabilistic relationships among the set of variables. This approach offers the underlying reasoning engine to support project management decisions providing a formal model that can be used by project managers to illustrate the effects of uncertainty on a software project management anti-pattern.

The suggested approach takes into consideration the characteristics and the needs of the individual software organization under assessment and does not demand a large amount of resources and investment costs. The method provides a generic Bayesian Network that models application development, which can be tailored to the needs of the development environment, applied. Bayesian analysis can make measurable each concept represented in the 4Ps of project management, People, Process, Project, Product. Bayesian Networks is an easily applied and comprehensible statistical tool that can provide very useful information to software managers. On the other hand the representation form of anti-patterns is a very descriptive and helpful tool in the hands of managers that can help them identify problems and provide easy and quick to launch re-factored solutions.

Acquiring a richer set of data from empirical investigations would be more helpful in creating knowledge from software development. A web-based community of software project management anti-pattern contributors would help in the direction of establishing a freely available, online knowledge base that could provide the tools to evaluate the impact of management decisions and decision support to software project managers worldwide.

References

1. Aurum, A., Jeffery, R., Wohlin, C., Handzic, M. (eds.): Managing Software Engineering Knowledge. Springer, Berlin (2003)
2. Bibi, S., Stamelos, I.: Software process modeling with bayesian belief networks. In: Online Proceedings of 10th International Software Metrics Symposium. Metrics (2004)
3. Bibi, S., Gerogiannis, V., Kakarontzas, G., Stamelos, I.: Ontology based Bayesian software process improvenent. ICSOFT EA **2014**, 568–575 (2014)
4. Boehm, B.W.: Software Engineering Economics, 1st edn. Prentice Hall PTR, NJ (1981)
5. Brown, W., McCormick, H., Thomas, S.: AntiPatterns in Project Management. Wiley, New York (2000)
6. Cheng, J.: Power constructor system. (1998). http://www.cs.ualberta.ca/jcheng/bnpc.htm
7. Dalcher, D., Thorbergsson, H., Benediktsson, O.: Comparison of software development life cycles: a multi project experiment. IEE Proc. Softw. Inst. Eng. Technol. **154**(3), 87–101 (2006)

8. Davenport, T., Prusak, L.: Working Knowledge How organizations Manage What They Know. Harvard Business School Press, Boston (2000)
9. Eloranta, V.-P., Koskimies, K., Mikkonen, T.: Exploring ScrumBut-an empirical study of Scrum anti-patterns. Inf. Softw. Technol. **74**, 194–203 (2016)
10. Fenton, N., Bieman, J.: Software Metrics: A Rigorous and Practical Approach. CRC press, Boca Raton (2014)
11. Fenton, N., Marsh, W., Neil, M., Cates, P., Forey, S., Tailor, M.: Making resource decisions for software projects. In: Proceedings of the 26th International Conference on Software Engineering (ICSE'04). pp. 397–406. (2004)
12. Hazrati, V.: Is five the optimal team size? (2009). http://www.infoq.com/news/2009/04/agile-optimal-team-size
13. Jensen, F.: Bayesian Networks and Decision Graphs. Springer, New York (2001)
14. Khodakarami, V., Abdi, A.: Project cost risk analysis: a Bayesian networks approach for modeling dependencies between cost items. Int. J. Proj. Manag. **32**(7), 1233–1245 (2014)
15. Laplante, P., Neil, C.: Antipatterns: Identification, Refactoring and Management. Taylor&Francis, Boca Raton (2006)
16. Lucia, D.A., Pompella, E., Stefanucci, S.: Assessing effort estimation models for corrective software maintenance through empirical studies. Inf. Softw. Technol. Elsevier **47**(1), 5–6 (2005)
17. Okutan, A., Yildiz, O.: Software defect prediction using Bayesian networks. Empir. Softw. Eng. **19**(1), 154–181 (2014)
18. Settas, D., Bibi, S., Sfetsos, P., Stamelos, I., Gerogiannis, V.: Using bayesian belief networks to model software project management antipatterns. In: 4th ACIS International Conference on Software Engineering Research, Management and Applications (SERA 2006). pp. 117–124. (2006)
19. Shepperd, M., Schofield, C., Kitchenham, B.: Effort estimation using analogy. In: 18th International Conference on Software Engineering (ICSE' 96). ACM (1996)
20. Silva, P., Moreno, A.M., Peters, L.: Software project management: learning from our mistakes [voice of evidence]. IEEE Softw. **32**(3), 40–43 (2015)
21. Stamelos, Ioannis: Software project management anti-patterns. J. Syst. Softw. **83**(1), 52–59 (2010)
22. Stamelos, I., Angelis, L., Dimou, P., Sakellaris, P.: On the use of bayesian belief networks for the prediction of software productivity. Inf. Softw. Technol. **45**(1), 51–60 (2003)
23. Terry, F., Wayne, S.: The effect of decision style on the use of a project management tool: an empirical laboratory study. DATA BASE Adv. Inf. Syst. **36**(2), 28–42 (2005)

Knowledge Engineering of System Refinement What We Learnt from Software Engineering

Rainer Knauf

Abstract Formal methods are a usual means to avoid errors or bugs in the development, adjustment and maintenance of both software and knowledge bases. This chapter provides a formal method to refine a knowledge base based on insides about its correctness derived from its use in practice. The objective of this refinement technique is to overcome particular invalidities revealed by the application of a case-oriented validation technology, i.e. it is some kind of "learning by examples". Approaches from AI or Data Mining to solve such problems are often not useful for a system refinement that aims at is an appropriate modeling of the domain knowledge in way humans would express that, too. Moreover, they often lead to a knowledge base which is difficult to interpret, because it is too far from a natural way to express domain knowledge. The refinement process presented here is characterized by (1) using human expertise that also is a product of the validation technique and (2) keeping as much as possible of the original humanmade knowledge base. At least the second principle is pretty much adopted from Software Engineering. This chapter provides a brief introduction to AI rule base refinement approaches so far as well as an introduction to a validation and refinement framework for rulebased systems. It also states some basic principles for system refinement, which are adopted from Software Engineering. The next section introduces a refinement approach based on these principles. Moreover, it considers this approach from the perspective of the principles. Finally, some more general conclusions for the development, employment, and refinement of complex systems are drawn. The developed technology covers five steps: (1) test case generation, (2) test case experimentation, (3) evaluation, (4) validity assessment, and (5) system refinement. These steps can be performed iteratively, where the process can be conducted again after the improvements have been made.

R. Knauf (✉)
Ilmenau University of Technology, PO Box 10056, 98684 Ilmenau, Germany
e-mail: rainer.knauf@tu-ilmenau.de

G.J. Nalepa and J. Baumeister (eds.), *Synergies Between Knowledge Engineering and Software Engineering*, Advances in Intelligent Systems and Computing 626, https://doi.org/10.1007/978-3-319-64161-4_5

1 Introduction

At a first glance, system validation aims at deriving some validity statements or a (lat or structured) metrics that says something about the degree of validity. However, validity statements that describe the particular weaknesses of a system may also serve as a launch pad for system refinement. So the final objectives of any validation technology is both (1) deriving indications for the system employment or rejection for the intended application and (2) refining the system towards overcoming the revealed invalidities.

Like Knowledge Acquisition, system refinement is a challenge to Knowledge Engineering. A typical way to face such problems is looking at approaches so far, but also at the way other (than AI) communities solved similar problems.

Here, a brief introduction to AI rule base refinement approaches so far is provided (Sect. 2) as well as an introduction to a validation and refinement framework for rule based systems (Sect. 3). This section also states some basic principles for system refinement, which are adopted from Software Engineering. Section 4 introduces a refinement approach based on these principles. After that, Sect. 5 considers this approach from the perspective of the principles.

Finally, in Sect. 6 some more general conclusions for the development, employment, and refinement of complex systems are drawn, which looks somewhat similar to principles applied in software engineering.

2 Refinement Approaches

The central purpose of refinement approaches is to adjust a system in a way that it overcomes revealed invalidities. In particular, they aim at a minimal refinement so that the falsified inputoutput behavior or the falsified trace through the system are fixed. There is a history of attempts to face this problem for rule based systems.

The classical rule refinement system is Teiresias [4]. It interactively guides a human validator through a faulty rule trace for the processed case knowledge. However, it is not able to determine the impact of refinements on other cases of the examined case base. To overcome this drawback, multiple case analysis systems like Seek/Seek2 [6, 9] have been developed. It gathers statistical performance information about all rules of the validated rule base in order to suggest the domain expert appropriate refinements. However, it does not validate the correctness of the refined intermediate reasoning traces. Ginsberg also published an approach called Reduced Theory Learning System (Rtls) [8]. Here, the term reduction means that a deep rule base is to be transformed into a flat one, whereas the reverse process is called retranslation. However, there is no exact retranslation procedure is known yet [8]. Interestingly, [7] claims that besides the retranslation problem his approach is only suitable for medium size rule bases. Moreover, according [12] not every rule base can be reduced. The performed retranslation step can not ensure that the refined rules are acceptable from the semantic point of view. Furthermore, structural anomalies can be introduced by this relaxed retranslation procedure [15]. Indeed, remaining anomalies are a strong indication for topical incorrectness with respect to the rules' interpretation. As a result, a rule base refined by such a technology is difficult to read and to interpret.

A relevant development is the tool Krust [14]. The refinement approach of this tool applies a hillclimbing search of an optimal refinement. It first generates several refined rule bases, which fix each incorrect example. After that, it chooses the refined rule base with the highest accuracy on the training examples. However, hill climbing procedures bag the risk to miss a real maximum and to end up with a local maximum instead. As a result, even after this refinement, there might be many cases, which are still processed in an invalid manner by the system.

Besides particular individual drawbacks, these approaches share the property that they can not produce a rule base which is 100 % correct.[1] Furthermore, they may cover some inherent anomalies and may not be interpretable by topical human experts. [5] introduces an interactive method, which consists in two steps that are performed in a loop until there is no further inconsistency found. It first checks the consistency and the completeness of and than, in second step, it performs the refinement interactively by involving human experts. This seems to be a reasonable idea, but it suffers from the need of human workload and from the fact that human experts may not able to analyze the consequences suggested changes on all the inference paths effected by it.

Also, [3] proposed an incremental way to refine knowledge bases. However, it aims at a step by step construction of a knowledge base than detecting and fixing inconsistencies in an existing knowledge base.

3 A Framework for Validation and Refinement

The developed technology covers five steps (see Fig. 1): (1) test case generation, (2) test case experimentation, (3) evaluation, (4) validity assessment, and (5) system refinement. These steps can be performed iteratively, where the process can be conducted again after the improvements have been made.

Here, the refinement step is focused. The preceding validity assessment step provides different validity measures according to their purpose: validities associated with outputs, rules, and test data as well as a global system's validity.

Based on these validities, the system refinement leads to a restructured rule base that maps the test case set exactly to the solution that obtained the best rating from the expert panel. Thus, the more the test case set is representative for the domain, the more the system refinement technology leads to a correct model of reality.

System refinement based on example cases has to be considered in the context of learning by examples. There are plenty of formal learning approaches that solve

[1]Correctness, in this context, means correctness w.r.t. a set of test cases.

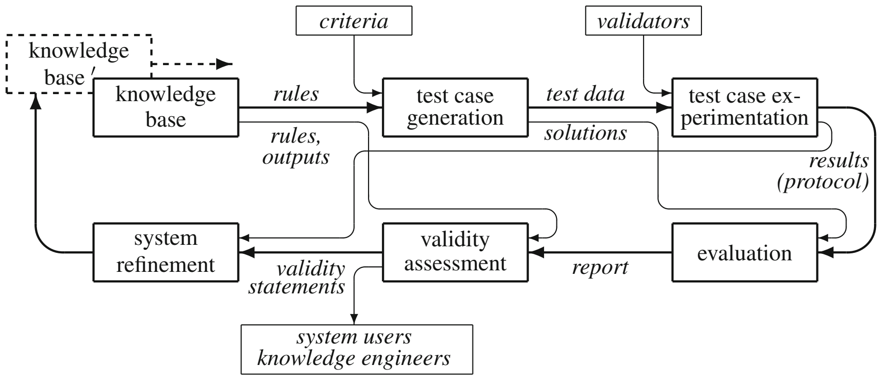


Fig. 1 Steps in the proposed validation process

tasks like this. Usually, they aim at developing rules that map test data with known solutions (examples) to their correct solution. They do not claim that all other cases are mapped correctly.

Unfortunately, some of the mentioned AI technologies to learn by examples generate rules that might reflect reality fairly well, but are not "readable" (or rather, interpretable) by domain experts. Even worse, they might construct rules that reflect the examples correctly, but are wrong with respect to the causal connection they express.

To avoid this drawback, system refinement techniques should be based upon two fundamental assumptions, which are well known from the way of considering the validation issue by the Software Engineering community [1, 2]:

1. It is assumed that the initial rule base was written with the intention of being correct, and if it is not correct, then a close variant of it is.
2. It is assumed that each component of the rule base appears there for some reason. Therefore, if a rule is invalid, it cannot just be removed. Rather, we have either

    a. to find out its reason for being in the rule base, and one (or several) alternative rule(s) that will satisfy that reason after the incorrect rule has been discarded or
    b. to develop ideas how to modify the rule with the objective of improvement.

Software engineers refer to these assumptions as the so called *competent programmer assumption*. In adoption of these principles, the present rule refinement technique tries to change the rule base as little as possible and to provide a close substitute for each piece of the knowledge base that will be removed.

The main idea of the refinement technique developed here is to find rules that are "guilty" in the system's invalidity and to replace them by rules, which map at least the example cases to their best rated solutions.

4 The Developed Refinement Strategy

The main idea of the technique is that the test case experimentation step revealed valuable knowledge about cases, for which the system failed: the human experts' solutions. Since these solutions are also a subject of the rating process (besides the system's solution), a humanmade solution might come out as being the very best one, the so called optimal solution.

The refinement strategy provided here is based on the following facts:

1. There is a knowledge base, which performs a mapping S from the input set I to the output set O: $S : I \to O$.
2. There are test data (input parts of the so called *Reasonable Set of Test Cases* $ReST$) $\Pi_{inp}(ReST)$ with both (2a) a system solution and (2b) a solution which is most heavily supported by the validation panel for each test data $t_j \in \Pi_{inp}(ReST)$.

If the "human made" solution (2b) is different from the system's one (2a), the system needs to be fixed.

Of course, system refinement has to be set into the context of learning by examples. Classical approaches like the Iterative Dichotomizer ID3 [13] for computing a classification rule set or concepts based on Anti-Unification [10] for computing a best inductive conclusion are not usable here. The basic difference is that we don't have just examples (test cases with solutions, which got "good marks" by experts) and look for rules, which explain them consistently. What we have is a (generally trustworthy) knowledge base and some "counter examples" that are mapped to a "bad solution" by it.

It is certainly the wrong way to construct a completely new knowledge base by means of the experimentation results. Of course, such a knowledge base maps all the performed cases to their best rated solution. On the other hand, (1) valuable former knowledge of the knowledge base would be wasted and (2) the computed knowledge base would be difficult to interpret for domain experts.

An invalidity occurs as a wrong system's output. Therefore, the technology begins with so called "last rules", i.e. rules that infer a final system's output. The procedure aims at the construction of one or more substitute(s) for this rule which infer the best rated output(s). In the first instance, the developed *if*-part(s) contain(s) just expressions about the system's input data. The following re-compiling mechanism utilizes the precompiled knowledge in the knowledge base by systematically substituting the expressions of the *if*-part by *then*-parts of other rules. The proposed technique consists of the following steps:

1. First, those rules that are guilty in the system's invalid behavior are discovered. Since the validation assessment step provides validity statements associated with (final) system's outputs, "guilty rules" are identified by considering rules that have final outputs as their *then*-parts. This is performed by an analysis of the ratings for the test cases, which used that rule. This step includes the identification of a so called "optimal solution" to each test data, i.e. a solution, which enjoys the best rating by the expert panel.

2. Next, the simple situation is considered that all test cases using a guilty rule have the same optimal solution. Here, the refinement consists in substituting this invalid *then*-part by this optimal solution.
3. Next, guilty rules with various optimal solutions for the test cases using the rule are considered. Here, a formal reduction system systematically constructs one or some new rule(s) as a substitute for the guilty rule.
4. The resulting new rules are "flat" i.e. they infer directly from system's inputs to system's outputs. To utilize the pre-compiled knowledge of the knowledge base, which occurs as rules with an intermediate hypothesis as their *then*- parts, the new rule(s) are re-compiled. Furthermore, the resulting knowledge base will be inspected for rules, which can never be used, because their *then*-part is an intermediate hypothesis, which is not needed after the rule base refinement. These rules are removed.

The last step does not change anything in the inputoutput behavior of the resulting rule base. However, it makes it more compact and easier to interpret by domain experts.

4.1 Finding "Guilty Rules"

All rules r_l, which infer a final solution sol_k, are the subject of the considerations. Based on the results of the previous steps of the validation framework, the following considerations apply:

1. Each r_l received an associated validity degree $v(r_l) = \frac{1}{|T_l|} \sum\limits_{[t_j, sol_k] \in T_l} v_{sys}(t_j)$. Here, T_l is the subset of test cases that used the rule r_l, $v_{sys}(tj)$ is a validity degree of the system for a test case t_j as computed in a previous step. $v_{sys}(tj)$ is a number that ranges between 0 (totally invalid) and 1 (totally valid) in the eyes of the validation panel.
2. There is a set T_l^* containing all test cases with test data parts occurring in T_l and all solution parts, which came up in the experimentation, regardless of whether the solution is given by an expert or the system: $T_l^* = T_l \cup \{[t_j, sol(e_i, t_j)] : \exists [t_j, sol_k] \in T_l\}$.
3. Next, the set T_l^* is split (partitioned) according to the different solution parts $sol_1, \ldots, sol_m$ of the test cases in T_l^*. This leads to m disjoint subsets $T_{li}^* \subseteq T_l^*$ $T_{l1}^*, \ldots, T_{lm}^*$ One of the subsets is the one collecting the test cases with the system's solution sol_k.
4. Analogously to the computation of the system solution's validity $v_{sys}(sol_k)$, a validity $v(r_l, sol_p)$ $(1 \leq p \leq m)$ of each solution sol_p can be computed:

$$v(r_l, sol_p) = \frac{1}{|T_{lp}^*|} \sum_{[t_j, sol_p] \in T_{lp}^*} \frac{1}{\sum\limits_{i=1}^{n} \left(cpt(e_i, t_j) \cdot c_{ijq}\right)} \cdot \sum_{i=1}^{n} \left(cpt(e_i, t_j) \cdot c_{ijq} \cdot r_{ijq}\right)$$

Here, the c_i and r_i are the certainties and ratings provided by the experts during the experimentation session and $cpt(e_i, t_j)$ is the estimated competence of the expert e_i for a test case t_j as proposed in [11].

5. There is an "optimal validity" of rule r_l, which is the maximum of all $v(r_l, sol_p)$ among all solutions sol_p occurring in T_l^*. The solution, to which this maximum is associated, is called the optimal solution sol_{opt} for r_l: $v_{opt}(r_l, sol_{opt}) = max(\{v(r_l, sol_1), \ldots, v(r_l, sol_m)\}$. $v_{opt}(r_l)$ can be considered an upper limit of the reachable rule–associated validity for rule r_l.
   There is a criteria based strategy [11] to identify the optimal validity in case several solutions received the same (and maximal) validity v_{opt}. In case $v_{opt}(rl, sol_{opt}) > v(r_l)$ there is at least one solution within T_l^*, which obtained better marks by the experts than the system's solution. In this case r_l is guilty and has to be modified: $v_{opt}(r_l, sol_{opt}) > v(r_l) \quad \Rightarrow \quad r_l$ is guilty.

As a result of applying this technology to each rule that concludes a final system's output, the guilty rules will be identified.

4.2 Simple Refinement by Conclusion Replacement

If for all test cases $t_j \in T_l$, for which the system used r_l, sol_s is the solution that met the maximum experts' approval, in rule r_l the conclusion part is substituted by sol_s and the rule r_l is exonerated: $\forall [t_j, sol_k] \in T_l$: sol_s is "optimal solution" for $[t_j, sol_k] \Rightarrow r_l$: $(if$ -part $\rightarrow sol_k) \hookrightarrow (if$ -part $\rightarrow sol_s)$.

4.3 Replacing the If-Part of the Remaining Guilty Rules

The remaining guilty rules are used by a set of test cases, which have different optimal solutions. The subsets with the same optimal solution are considered separately:

1. T_l of the rule r_l is split into subsets T_l^s $(1 \leq s \leq n)$ according to the solution sol_s for each t_j that obtained the highest validity $v(r_l, sol_s)$. The new *if*-part(s) of the new rule(s) instead of r_l are expressions $e_i \in E$ of a set of p new alternative rules $r_l^1, r_l^2, \ldots, r_l^p$ for each T_l^s and will be noted as a set of sets $P_l^s = \{\{e_1^1, \ldots, e_{p_1}^1\}, \ldots, \{e_1^p, ldots, e_{p_p}^p\}\}$. The corresponding rule set of P_l^s is $r_l^1 : \bigwedge_{i=1}^{p_1} e_i^1 \rightarrow sol_s, \ldots, r_l^p : \bigwedge i = 1^{p_p} e_i^p \rightarrow sol_s$.
2. Pos is the set of Positions (dimensions of the input space), at which the input data $t_j \in \Pi_{inp}(T_l^s)$ of the test cases $t_j \in T_l^s$ are **not** identical. The generation of the *if*-parts P_l^s is managed by a formal *reduction system*, which is applied to Triples $[T_l^s, Pos, P_l^s]$ until Pos becomes the empty set $\emptyset$.
3. The initial situation before applying the reduction rules is $[T_l^s, Pos, P_l^s]$ with $P_l^s = \{\{(s_1 = s_1^{ident}), \ldots, (s_q = s_q^{ident})\}\}$. $s_1, \ldots, s_q$ are those positions where all

test data $t_j \in \Pi_{inp}(T_l^s)$ have the same (identical) value s_i^{ident} and again, Pos is the set of the remaining positions.

4. The reduction terminates, if the situation $[T_l^s, \emptyset, P_l^s]$ is reached.

The reduction rules that are applied to these Triples are shown in Table 1.

Depending on what kind of dimension the next position $pos \in Pos$ is, one of the rules $R1$ or $R2$ can be applied:

- $R1$ handles the case that the considered position pos refers to dimension s_{pos}, which is enumerable, has a finite set of values, and there is no domain-related ordering relation between its values, and

Table 1 Reduction rules to construct better rules systematically

Reduction rules

R1 • $pos \in Pos, s_{pos}$ has a finite value set with no well-defined $\leq$ relation

• $\{s_{pos}^1, \ldots, s_{pos}^m\}$ are the values of s_{pos} occurring in T_l^s $\Rightarrow$

$$[T_l^s, Pos, \{p_1, \ldots, p_n\}] \quad \hookrightarrow$$

$$1. \quad [T_l^{s,1} \setminus \{[t_j, sol_s] \in T_l^s : s_{pos} \neq s_{pos}^1\}, Pos \setminus \{pos\}, \bigcup_{i=1}^{n} p_i \cup \{(s_{pos} = s_{pos}^1)\}]$$

$$2. \quad [T_l^{s,2} \setminus \{[t_j, sol_s] \in T_l^s : s_{pos} \neq s_{pos}^2\}, Pos \setminus \{pos\}, \bigcup_{i=1}^{n} p_i \cup \{(s_{pos} = s_{pos}^2)\}]$$

$$\bullet\bullet\bullet$$

$$\text{m.} \quad [T_l^{s,m} \setminus \{[t_j, sol_s] \in T_l^s : s_{pos} \neq s_{pos}^m\}, Pos \setminus \{pos\}, \bigcup_{i=1}^{n} p_i \cup \{(s_{pos} = s_{pos}^m)\}]$$

Continue with each $T_l^{s,i}$ $(1 \leq i \leq m)$ separately.

R2 • $pos \in Pos, s_{pos}$ has a value set with a well-defined $\leq$-relation

• s_{pos}^{min} is the smallest value of s_{pos} within T_l^s

• s_{pos}^{max} is the largest value of s_{pos} within T_l^s $\Rightarrow$

$$[T_l^s, Pos, \{p_1, \ldots, p_n\}] \quad \hookrightarrow$$

$$[T_l^s, Pos \setminus \{pos\}, \bigcup_{i=1}^{n} p_i \cup \{(s_{pos} \geq s_{pos}^{min}), (s_{pos} \leq s_{pos}^{max})\} \cup S_{excl}]$$

S_{excl} is the set of excluded values for s_{pos}, which have to mapped to a solution different from sol_s because of belonging to some other T_u^v with $v \neq s$:

$$S_{excl} = \{\ (s_{pos} \neq s_{pos}^j) :\ \exists [t_j, sol_s] \in T_l^s\ \exists [t_m, sol_v] \in T_u^v (v \neq s) \text{ with}$$
$$\forall p \neq pos\ ((s_p^j = s_p^m) \text{ and } (s_{pos}^{min} < s_{pos}^m < s_{pos}^{max}))\ \}$$

- $R2$ handles the case that the considered position pos refers to dimension s_{pos}, which is numerical respectively it has a (finite or infinite) set of values, and there is a domain-related ordering relation $\leq$ between its values.

The binary case is a special case of the first one, in which s_{pos} has exactly two different values. Here, $\{s_p^1 os, s_p^2 os\}$ are the values of s_{pos} occurring in T_l^s with $s_{pos}^1 = true$ and $s_{pos}^2 = false$.

If $R1$ is applicable, the reconstruction system produces m new triple $[T_l^{s,i}, Pos \setminus \{pos\}, P_l^{s,i}]$ $(i = 1, \ldots, m)$ from $[T_l^s, Pos, \{p_1, \ldots, p_n\}]$, one for each value s_{pos}^i occurring at the position pos in the test data in T_l^s. Each of the new triples differs from the original triple in (1) a test case set $T_l^{s,i} \subseteq T_l^s$, which only contains the test cases with the considered value s_{pos}^i at position pos and (2) a set $P_l^{s,i} \supseteq P_l^s$, which is enlarged by the expression $(s_{pos} = s_{pos}^i)$. Furthermore, (3) the position pos is removed from the set of remaining positions.

If $R2$ is applicable, each element P_l^s has to be extended by a condition, which describes the range of s_{pos} within the set of cases in T_l^s. This range is put up by its minimum value s_{pos}^{min} and its maximum value s_{pos}^{max} within T_l^s. However, it may happen that this range is not "homogeneous" with respect to the optimal solution, i.e. there are counterexamples in some other T_u^v, which have (1) at position pos a value between s_{pos}^{min} and s_{pos}^{max}, (2) at all the other positions values described by the rest of the actually extended $p_i \in P_l^s$, i.e. by the future *if*-part of the upcoming rule for solution sol_k, and (3) have to be mapped to a different final solution sol_v, i.e. $k \neq v$. These counter-examples have to be excluded by expanding each p_i with a corresponding exclusion set S_{excl}. In S_{excl} the negation of each value of any s_i is described.

The case that a counterexample belongs to a T_u^s (i.e. to a different rule r_u for the same solution sol_k) does not have to be considered individually. It does not matter, which way, i.e. by using which rules, the future knowledge base comes up with the better solution. It is just important that it finds the better solution after refinement. In the latter case, we produce a non-determinism in the knowledge base with the same final end, i.e. a situation, where two different rules are applicable, but both end up in the same system's solution.

The entire reduction system is applied in both (1) recursive loops (by using rule $R1$) and (2) iterative loops (by using rule $R1$ or $R2$, because the entire rule system is applied as long as $Pos \neq \emptyset$)just like a system of production rules. Within each cycle (1) the applicable rule is identified (exactly one), (2) the rule is applied, which leads to a larger set P_l^s and a smaller set Pos, and (3) the same business starts again until each position is considered, i.e. $Pos = \emptyset$. It is quite obvious that the reduction system is terminating, complete, and correct.

However, for cases other than the test cases, this technique can't guarantee correctness, of course. This is due to the nature of all test case approaches. But for typical AI application fields there is no other reasonable way of validation than using "good" test cases and examining them as objective as possible. More can't be done.

4.4 Recompiling the New Rules and Removing the Unused Rules

The new rules generated so far are “one-shot-rules”, i.e. they infer directly from a system’s input to a system’s output. These rules might be difficult to read, because they may have very long if parts, and difficult to interpret by subject matter experts. This problem can be defused by introducing the intermediate hypotheses into the computed new rules. Here, we consider two cases:

1. First, if the *if* -part of a new rule contains a subset of expressions that is the complete if -part of another rule having an intermediate solution as its then-part, this subset is replaced by the corresponding intermediate solution:

   $$\exists r_i : (if - part_1 \rightarrow int_1) \exists r_j : (if - part_1 \wedge if - part_2 \rightarrow int - or - sol) \Rightarrow (if - part_1 \wedge if - part_2 \rightarrow int - or - sol) \hookrightarrow (int_1 \wedge if - part_2 \rightarrow int - or - sol)$$

2. Second, we remove rules that having an intermediate hypothesis as its *then*-part, which is not used in any *if* -part of any rule:
   $\exists r_i : (if - part_1 \rightarrow int_1) \neg \exists r_j : (int_1 \wedge if - part_2 \rightarrow int - or - sol) \Rightarrow r_i : (if - part_1 \rightarrow int_1) \hookrightarrow \emptyset$

5 The Technique in the Context of the Assumptions

The technique as described in the previous chapter follows the ideas of Ackermann et al. [1] and Adrion et al. [2] that are originally developed for use in classical software validation and mentioned here as an introduction.

Whenever a rule is indicated as “guilty” in some invalid system behavior, it will be changed as slightly as it can be to map the examples consistently. The basic idea behind the approach is to keep this rule and change it in a manner that the counterexamples, i.e. the test cases that have been solved incorrectly by the considered rule, will be excluded from using this rule.

To handle these counterexamples, some extra rules will be computed, which map these test cases to their correct solution. In the context of validation, we can define “correct” just by “There is no different manmade solution that obtained better marks by the validation panel.”. More can’t be done.

Since the only hint about the present kind of invalidity is some better final solution to the examined test cases, the rule reconstructing technique focuses on rules that have final solutions as their conclusion parts.

In a first setting, the upcoming new rules infer directly from the system’s inputs to the system’s outputs, i.e. they are “one shot” rules. On one hand, these rules secure a correct inputoutput behavior of the system. On the other hand, such rules tend to be nonreadable and noninterpretable in the context of the rest of the knowledge base. Usually, they tend to have many expressions in their condition parts, i.e. they are very long.

To avoid this drawback and to minimize the number of rules by utilizing existing rules as precompiled knowledge, the computed rules are adapted to fit in the context of the rest of the rule base by using their conclusionparts within the condition part of the upcoming rules.

In particular, if there is at least one test case that is performed correctly by this rule, a very close variant of the original rule remains in the system. This variant is characterized by having the same conclusion part and a slightly changed (usually longer) condition part. These changes are due to the fact that the counter-examples have to be excluded.

All the remaining knowledge (besides the rules that handle these counter- examples) is not touched at all by the rule reconstruction technique. Even the invalid rules are only changed as less as possible. The only objective of the refinement strategy is to handle the counter-examples in a way that is indicated to be more valid by the validation technology. The other examples, which have been handled in a valid manner by this “guilty” rule will still by mapped to the same solution as before. The basic message behind is:

If you do not know anything better by practical experience, believe in the knowledge provided so far and don’t touch this knowledge.

Since the knowledge base so far is a product of human’s experience (either the authors’ original knowledge or an improved version of it by applying this technology), there is even a more general message behind:

If you do not know anything better by practical experience, believe in the knowledge provided by other humans, i.e. inherit their experience.

To summarize, the presented technique keeps as much as it can from each rule. This is performed by

1. reconstructing the rules in a manner such that they handle the counter-examples (and only them) differently from the original rule base[2] and
2. using as much as it can from the original rule base by “compiling” the new rules together with the original ones in the knowledge base.

The latter issue utilizes correlations between the former (human-made) and the upcoming pieces of knowledge, which are artificially computed, but based on human insights within the validation process.

The author truly believes that this is the right thing to do, because it sets the new knowledge in the context with the original one. This way, the modified rule base should be more easily to be interpreted by human experts (who provided the original knowledge) and additionally, the knowledge base will be optimal with respect to its size, i.e. both the number of rules and their complexity.

[2]Without effecting the basic message of the present paper, the author should “admit”, that this is a simplified description of the truth. Especially in case of non-discrete input data the situation occurs slightly more complicated. For details, see [11].

6 Conclusion

The author feels that the presented technique is a general way to refine AI systems in particular, and technical systems that contain knowledge in general. Engineers who develop and refine any technical system often say "Never change a running system.". Here, we extend this point of view by postulating the following three general issues:

1. Don't change a system that works well.
2. Do change a system that works almost well as slightly as you can. Keep as much as you can to avoid the risk making things worse and just change those parts that handle the particular invalidities.
3. Do not try to rene a system that is invalid for many cases or for extraordinary important cases (safety critical ones, for example). Here, some thought to rebuild either the system or even the general approach should be invested.

The refinement strategy presented here falls into the second class, i.e. it provides a formal method to "repair" a working system with the aim of handling singular invalid (test) cases correct in future and to keep the former input/output behavior for all other cases.

Since the test cases can be considered as examples, i.e. as input/output pairs with a known correct output,[3] the strategy should also be considered in the context of "learning by examples". Classical AI technologies to perform tasks like these (ID3 [13], for example) aim at producing a rule set that classifies the examples correctly. On the one hand, they enjoy consistency with the examples, but on the other hand, they suffer from being somehow "artificially constructed" and are not interpretable by domain experts.

Moreover, the risk that such rules reflect the reality wrong is much higher than by using techniques such as the one presented here. This is because the "non-examples" (i.e. all test data that can occur in practice but are not a member of the example set respectively the test case set, in our approach) are used to optimize the upcoming rule set. The inputs that are not mentioned in any example (that are not a test case, in our setting) are mapped to any output that is "useful" with respect to some optimization issue (the number or the length of the constructed rules, for example).

The presented technique, on the other hand, is based on the assumption, that all "non-examples" are handled correctly by the former knowledge base. It doesn't change the behavior for cases that have not been examined as test cases within the validation technology. Since the historic knowledge base is a product of human thought, the probability that these "non-examples" are handled correctly is much higher.

To sum up, the well known competent programmer assumption in Software Engineering has been adopted in Knowledge Engineering as a competent Knowledge Engineer assumption when refining AI systems.

[3]Again, correctness here means validity and is nothing more and nothing less than a behavior that obtained "good marks" by some (human) validation panel.

References

1. Ackermann, A.F., Fowler, P.J., Ebenau, R.G.: Software Validation. Elsevier, Amsterdam (1984)
2. Adrion, W., Branstadt, M., Cherniavsky, J.: Validation, verification and testing of computer software. ACM Computing Surveys, pp. 159–182 (1982)
3. Cao, T.M., Compton, P.: A consistency-based approach to knowledge base refinement. In: Proceedings of the 18th International Florida Artificial Intelligence Research Society Conference, pp. 221–225 (2005)
4. Davis, R., Lenat, D.B.: Knowledge Based Systems in Artificial Intelligence. McGraw Hill Int. Book Company, New York (1982)
5. Djelouah, R., Duval, B., Loiseau, S.: Interactive refinement of a knowledge base. In: Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society Conference, pp. 325–330 (2004)
6. Ginsberg, A.: Automatic Refinement of Expert System Knowledge Bases. Pitman Publishing, London (1988)
7. Ginsberg, A.: Theory revision via prior operationalization. In: Proceedings of the 7th National Conference on Articial Intelligence (AAAI-88), pp. 590–595 (1988)
8. Ginsberg, A.: Theory reduction, theory revision, and retranslation. In: Proceedings of the 9th National Conference on Artificial Intelligence (AAAI-90), Boston, MA, pp. 777–782 (1990)
9. Ginsberg, A., Weiss, S.M., Politakis, P.: Automatic knowledge base refinement for classification systems. Artif. Intell. **35**, 197–226 (1988)
10. Knauf, R.: Inferenzmethoden. Script and slights to a course on Inference Methods, available (in German). http://www.tu-ilmenau.de/ki/lehre/
11. Knauf, R.: Validating rule-based systems. A complete methodology. Aachen: Shaker, Berichte aus der Informatik, Ilmenau, Technische Universitt, Habilitationsschrift (Habilitation Thesis, in German) (2000). ISBN 3-8265-8293-4
12. Liang, C.X.F., Valtorta, M.: Refinement of uncertain rule bases via reduction. Int. J. Approx. Reason. **13**, 95–126 (1995)
13. Quinlan, J.: Learning effcient classification procedures and their application to chess end games. Michalsky et al. (eds.): Machine Learning: An Artificial Intelligence Approach, Palo Alto, CA Tioga Publishing Corp. (1983)
14. Wiratunga, N., Susan Craw, S.: Incorporating backtracking in knowledge refinement. In: Vermesan / Coenen (eds): Validation and Verification of Knowledge Based Systems - Theory, Tools and Practice. Proceedings of the 5th European Conference on Validation and Verication (EUROVAV 99), Boston, MA Kluwer Academic Publishers, pp. 193–205 (1999)
15. Zlatareva, N.P., Preece, A.D.: State of the art in automated validation of knowledge based systems. Expert Syst. Appl. **7**, 151–167 (1994)

Using the Event-B Formal Method and the Rodin Framework for Verification the Knowledge Base of an Rule-Based Expert System

Marius Brezovan and Costin Badica

Abstract Verification and validation of a knowledge base of an expert systems are distinct activities that allow to increase the quality and reliability of these systems. While validation ensures the compliance of a developed knowledge base with the initial requirements, the verification ensures that the knowledge base is logically consistent. Our work is focused on the verification activity, which is a difficult task that mainly consists in determination of potential structural errors of the knowledge base. More exactly, we aimed to study the consistency of knowledge bases of rule-based expert systems that use the forward chaining inference, a very important aspect in the verification activity, among others, such as completeness and correctness. We use Event-B as a modelling language because it has a mathematical background that allows to model a dynamic system by specifying its static and dynamic properties. In addition we use the Rodin platform, a support tool for Event-B, which allows to verify the correctness of the specified systems and its properties. For a better understanding of our method, an example written in the CLIPS language is presented in the paper.

1 Introduction

Rule-based expert systems represent computer programs that are designated to solve problems specific to a certain domain in a manner similar to an human expert in that domain. The *knowledge base* (KB) of such as expert system (ES) is represented by a set of production rules, each *production rule* can be described syntactically as follows:

$$\text{If}\langle premise\rangle\text{Then}\langle conclusion\rangle,$$

M. Brezovan (✉) · C. Badica
University of Craiova, Craiova, Romania
e-mail: mbrezovan@software.ucv.ro

C. Badica
e-mail: cbadica@software.ucv.ro

G.J. Nalepa and J. Baumeister (eds.), *Synergies Between Knowledge Engineering and Software Engineering*, Advances in Intelligent Systems and Computing 626, https://doi.org/10.1007/978-3-319-64161-4_6

where $\langle premise \rangle$ is a sequence of logical conditions, while $\langle conclusion \rangle$ represents a sequence of some predefined actions.

Because of their efficiency, the market of rule-based expert systems has grown starting to 1985, when many expert systems have been developed. Unfortunately in the last two decades this market has diminished. In addition, today there is a gap between the production of rule-based expert systems and the research in this area: only a few companies still develop rule-based expert systems, although the research related to knowledge engineering is still active. This phenomenon is difficult to be investigated, and it is outside of the goal of the paper. However, in our opinion, one important reason for this trend is the significant amount of time for assembling and validating a knowledge base, which increases the cost of a such rule-based expert system.

While the *assembling* phase of a knowledge base is a relative standard activity, which is proven in terms of technology and procedures, the *validation* phase of a knowledge base has some drawbacks: (a) it is a time-consuming activity, and, more important, (b) there are not standard methodologies used in this activity. Moreover, the validation of a knowledge base is still a subject in the research area of rule-based expert systems.

1.1 Knowledge Engineering Versus Software Engineering

Because *validation* is a term used in two different and somewhat similar domains, *Knowledge engineering* (KE) and *Software engineering* (SE), it is important to highlight the similarities and the differences between the main terms from these domains. Both of these notions represent engineering disciplines that are concerned in all aspects of: (a) software production, in the case of SE, or (b) knowledge base development, in the case of KE. As a remark, the notion of KE has changed slightly in recent years, because today *knowledge* is related not only to knowledge-based systems, but also to web-based systems, and other information systems. We use however the above notion, because our concern in this paper are knowledge-based systems. In the following we refer the term of *knowledge base engineering* (KBE) instead of KE.

In the context of SE, software validation is defined as, *determination of the correctness of a program with respect to the user needs and requirements* [1]. Because of differences between building expert systems (ES) and conventional software programs, some authors consider that ES validation, and in particular KB validation, is somehow different from conventional software validation. We agree to Meseguer opinion [2], which claims that the definition of validation in the context of SE is also applicable in the context of KBE for ES and KB validation.

However, in the domain of KB validation terms like *validation*, *verification*, *testing* or *evaluation* are very frequently used, while their exact definitions are still unclear. According to [3], we consider *validation* as a global term that includes all others terms as specific aspects. More specific, the *validation* activity can be considered a composition of two different and complementary tasks:

- *verification*, which represents those activities that intend to verify the structural correctness of the KB content,
- *evaluation*, which represents those activities that intend to demonstrate the KB ability to generate correct conclusions.

While the meaning of *evaluation* is similar in SE and KBE, there is a difference regarding *validation methods* in these domains, because of the impossibility of a complete translation of user requirements into specifications [4]. Methods used in the evaluation tasks from KBE are very similar to the corresponding methods from SE, where the main activities are related to testing methodology. On contrary, methods used in the verification tasks from KBE are different to the corresponding methods from SE. Basically, these methods are based on the transformation of a KB into some specific information structures and then on the using of appropriate procedures for checking the structural correctness of the KB.

One of the techniques used both in SE and KBE in several phases of software development and knowledge base construction is related to *formal methods*. *Formal methods* are mathematically-based approaches to define a formal model, and this model can be then formally analyzed and used as a basis for a formal system specification, development and validation [5]. A review of using formal methods in KBE is presented in [6]. However, while in present there exists many formal methods, in KBE these methods are used more in the validation phase than in other phases of KBE, such as specification, model construction, or KB creation. Moreover, even in the validation phase there are few approaches of using formal methods to verify user requirements [7].

1.2 Related Work

KB validation means to determine whether or not the content of the KB accurately represents the knowledge of the human experts that supplied it. Despite the fact that for KB validation, a number of techniques similar to those of the SE domain are available, in this paper we are not concerned with this task.

KB verification determines whether KBs are flawed or not from the structural correctness point of view, requiring to check several knowledge properties, such as correctness, consistency and completeness [8]. This task is difficult to be performed in practice because of the large number of KB objects that can interact during the ES execution. In addition, while validation can be subjective, verification is mainly an objective test, because there exists several measures of the correctness of a KB.

In this paper we focus on the verification process because the verification process is more complex than validation process, and it is crucial for a correct functionality of rule-based systems. More precisely we approach the problem of consistency of rule bases of expert systems.

For detecting inconsistencies in knowledge bases, several methods have been proposed, most of them before 2000:

- *Petri Nets-based methods* that transform a rule base into a Petri net [9–11] and then use appropriate techniques for analysing Petri nets for detecting inconsistencies in KBs. Each production rule from a KB is translated to a transition in the resulted Petri net, input places representing conditions of the rule antecedent, and output places the actions of the rule consequent. However, the majority of Petri nets-based approaches are related to KBs that do not contain variables inside rules. Only few approaches (e.g. [11]) treat the case of rules containing variables. However, there exists an inconvenient in all these approaches, which increase the time for performing the verification task: all the possible initial states must be tested in order to assure the absence of inconsistencies.
- *Graph-based methods* that represent conceptual dependencies between rules and provide analysis techniques to study some rule properties, such as connectivity or reachability [12, 13]. Similar to the case of Petri nets-base approaches, graph-based approaches have the same inconvenient, because all possible values for the initial fact base must be tested in order to assure the absence of inconsistencies.
- *Algebraic methods*, where a knowledge base is first transformed into an algebraic structure, and then appropriate techniques are applied for the verification of the properties of the knowledge base [14, 15]. However there are few algebraic approaches for detecting inconsistencies in KBs. These approaches have a disadvantage similar to the disadvantage of Petri nets-based approaches: they can be applied only to KBs that do not contain variables inside rules. There are no approaches that treat the case of rules containing variables.
- *Declarative approaches* are new approaches that are used in KBs containing both rules and ontologies, when the knowledge base is transformed in Datalog facts and rules, and then programs for implementing the anomaly checkers are developed [16, 17].

1.3 Our Approach

According to [8], the main properties used for structural verification of a KB include *consistency*, *completeness*, and *correctness*. Our method treats only the problem of consistency of KBs of rule-based expert systems.

Our proposal uses a formal method for the verification of a rule base (RB) of an ES based on forward chaining inference, which contains variables inside rules. We use both the *Event-B* formal language [18] and its tool support called *Rodin* [18] for performing this task. As in other proposals, our method mainly consists in two steps:

- The *modelling phase* that translate the knowledge base using the *Event-B* language into an equivalent Event-B specification,
- The *verification phase* that uses the semantics of Event-B, and the *ProB* plug-in for the *Rodin* platform [19] for performing two different sub-tasks:
  - the model-checking, which is automated performed by the *ProB* plug-in,

– checking the consistency of the translated Event-B model by using the provers of *ProB* on the selected proof obligations.

Because each Event-B model consists of two parts:

- a *static part*, represented by one ore more *contexts*, where the data part of the model is specified,
- a *dynamic part*, represented by one or more *machines*, where the system evolution is described,

our method presents some *patterns* for translating the KB of a rule-based ES into an equivalent Event-B model:

- the *fact base* is used for generating the static part of the model,
- the *rule base* is used for generating the dynamic part of the model, where rules are translated to *events* of the corresponding machine.

There are several advantages of our method over other approaches:

- The *Event-B* method allows an easy modeling of a KB, as in the case of *Petri nets* approaches. Each rule from a RB can be easily modeled as an event in the machine of the translated Event-B model. In addition, the facts from the fact base, can be modeled as global variables in the same Event-B machine. Moreover, the Event-B language allows to model KBs that contain variables inside rules.
- In contrast to other approaches, a part of checking the consistency of a KB can be automatically realized by using the *ProB* for the *Rodin* platform, which is an animator and model checker for Event-B. In this way, the time used for the verification of a KB can be shortened, which is a goal for the development of rule-based systems.
- Using the *refinement* action of the *Event-B* method, the development of a KB can be done incrementally, in a similar way to the *agile development* process form the domain of software engineering.

In addition, by using the *Event-B* language, the inference engine of a rule-based expert system can be also easy modeled. In this way, an entire rule-based expert system can be modeled and tested before its implementation, not only its KB part. This is another way to shorten the time for the development of a rule-based expert system.

1.4 Overview of the Paper

The rest of this paper is organized as follows. Section 2 presents the elements of the *Event-B* language and its associated open framework *Rodin*. Section 3 presents the first step of our method that translates a KB into an equivalent Event-B model. In this Section there are presented some patterns that can be used in this translation, as well an example of translating a KB written in the CLIPS language into an Event-B model.

Section 4 presents the second step of the method: verification of the consistency of the translated Event-B model. Methods for verifying both static and dynamic properties related to the consistency of the model are presented. Conclusions and future work are presented in Sect. 5.

2 Event-B and Rodin

A wide range of *formal specification languages* have been proposed that can be used in different system and software development phases. Based on their mathematical background, these languages can be classified as: (a) *Predicate logic-based languages*, such as *Z* [20], *Object-Z* [21], *VDM* [22], *B* [23], and *Event-B* [24], (b) *Temporal logic-based languages*, such as *CTL* (Computational Tree Logic) [25] and *TLA* (Temporal Logic of Actions) [26], (c) *Process algebras*, such as *CSP* (Communicating Sequential Processes) [27] and *CCS* (Calculus of Communicating Systems) [28], and (d) *Algebraic Specifications*, such as *CASL* [29] and *OBJ* [30].

2.1 Event-B

Event-B [24] is a is a state-based formal method used for modeling discrete state transition systems, which uses *states* and *transitions* for defining the state of a modelled system and its evolution. States of a system are defined by *variables* and state changes are defined by *events*. Event-B has evolved from the classical *B* method [23].

However, the B Method is related more to the software development that can be used in the software engineering domain, while Event-B is related to the system development, and it can be used in a more general domain of system engineering.

An Event-B model contains two basic components:

- The *static part* of the model represented by one ore more *contexts*. Each context may contain *carrier sets*, s, which are names of data types used by the model, *constants* c, which correspond to simple mathematical objects, such as sets, binary relations, functions, numbers, etc., *axioms*, $A(c, s)$, which define the semantics of the carrier sets, and *theorems*, $TC(c, s)$, which specify some properties related to constants and carrier sets. During the development of a system, a context can *extend* one or more contexts by declaring additional carrier sets, constants, axioms or theorems.
- The *dynamic part* of the model, represented by *machines*, which describe the system evolution. Each machine may contain *variables*, v, whose current values define the current state of the system, *invariants*, $I(c, s, v)$, which define the valid states of the system, *theorems*, $TM(c, s, v)$, which specify some properties related to variables, carrier sets and constants, and a set of *events*, E, each event describing a transition of the system between two states.

An event, e, is an atomic specialized B operation that describes a state change:

$e \mathrel{\widehat{=}}$ any p where G(p, c, s, v) then S(p, c, s, v) end,

where p represents the sequence of parameters for the event e, $G(p, c, s, v)$ represents its enabling condition called *guard*, which is a conjunction of one or more predicates, and $S(p, c, s, v)$ is the action associated to the event that can change the value of variables v. The guard $G(p, c, s, v)$ specifies the necessary condition for occurring the event, and the action $S(p, c, s, v)$ describes the modification of the state variables when the event occurs. Denoting by v the state of the system before the occurrence of the event v, and by v' the state after its occurrence, the action S can be viewed as a *before-after* predicate that defines a relation between these states. It can be also described as $S(p, c, s, v, v')$. If several events are simultaneously enabled, then at most one of them can be executed. The choice of event to be executed is non-deterministic.

The set E of events contains *Initialisation* as a special event, $R_I(c, s, v')$, which describes the initial state of the system, by computing initial values of the variables.

Remark For simplicity, when possible, the constants c and carrier sets s from the static part of a model are not specified in the above predicates.

Event-B defines several *proof obligations*, mathematical formulas borrowed from the CSP language [27], which must be proved to show that a formal model fulfil its specified properties. The proof obligations are then discharged using certain inference rules.

Refinement is the only operation that can be applied to a machine, which represents a mechanism for introducing details about the dynamic properties of a model. If a machine N refines another machine M, then M is called the *abstract machine*, while N is a *concrete machine*.

2.2 Rodin

Rodin [18] is a software tool support for Event-B, which represents an open framework developed on the top of the Eclipse platform. This framework was developed as part of three successive European Union ICT Projects: *RODIN* (from 2004 to 2007) [18], *DEPLOY* (from 2008 to 2012) [31], and *ADVANCE* [32] (from 2011 to 2014).

The Rodin tool chain allows development and verification of Event-B models, ant it consists of three major components: the static checker, the proof obligation generator, and the proof obligation manager. The static checker performs syntactical verification of Event-B models, the proof obligation generator generates proof obligations needed to be proven in order to verify the correctness of the modelled system, while the goal of the proof obligation manager is to maintain the proof status and the proofs associated with the obligations.

The Rodin open platform is different from others formal methods tools, because it allows multiple parties to integrate their tools as plug-ins to support a rigorous development of Event-B models. Among other plug-ins developed for Rodin, *ProB* and *Theory* are two important plug-ins that complement the existing modelling and proof functionality of both the core Rodin platform and the Event-B mathematical language.

ProB [19, 33] is an animator and model-checker for both B and Event-B methods. The main goal of ProB is to check the consistency of a B (or Event-B) model. Checking the consistency of a model means to check whether the invariant of a B (or Event-B) machine is satisfied in all initial states, and whether its invariant is preserved by the operations (or events) of the machine. This task is performed by computing the state space of the model and then by performing an exhaustive search. ProB can also be used as an animator of a B (or Event-B) model. So, this plug-in represents not only a verification tool, but also a sophisticated debugging and testing tool.

Classical models in Event-B are specified by using two types of components: contexts and machines. The newer releases of Rodin define a new component called *theory* [34]. A theory is a way by which the Event-B mathematical language and its automated provers can be extended, independent of any particular model. *Language extensions* are added to the Event-B mathematical language in the form of datatypes, operators, and axiomatic definitions. *Proof extensions* are added to the Rodin core platform in the form of rewrite rules, inference rules, and polymorphic theorems. The theory component has been introduced in the *DEPLOY* ICT project [31].

3 Translatintg the Knowledge Base to an Event-B Model

One of the main advantages of our approach over other approaches is the fact that it allows to verify rule-based systems that use *variables*. As a consequence we model separately the two components of a knowledge base: the *fact base*, and the *rule base*.

3.1 A Specification Language for a Rule-Bases Expert System

For the sake of generality, we use a simple and generic language for describing rule-based system, which is close to the *CLIPS* [35] and *Jess* [36] languages. Because in a rule-based system, knowledge is represented by a set of rules and by a set of facts about the current situation, the generic rule-based language contains distinct syntactic constructions for these elements.

Facts are represented by a sequence of constants, where the first constant is a symbolic constant specifying the name of the fact, followed by a sequence of ordered pairs, each pair representing the name of an attribute, and its corresponding value:

$$\langle fact\rangle \ ::= \ '(' \ \langle \textit{fact–name}\rangle \ (\langle attribute\rangle \ \langle value\rangle)^{+} \ ')'.$$

The syntax of facts is similar to the syntax of *ordered facts* from CLIPS and Jess, but its meaning is closer to the *unordered facts*. A value can be a *single-value*, which is a constant from an appropriate set, or a *multi-value*, which is a sequence of constants from a given set. For example, a complex number specified by its real and imaginary parts, can be represented as $(complex\ re\ 7.2\ im\ 3.5)$, while a fact representing a list of numbers can be represented as (*list elems* 2 1 3 4 3).

For specifying attributes of a fact and their associated values, our generic language allows to define a *template* for each category of facts:

$$\begin{array}{l} \langle \textit{fact–template} \rangle\ ::=\ '('\ \langle \textit{fact–name} \rangle\ '('(\langle \textit{attr–name} \rangle\ \langle \textit{attr–type} \rangle\ \langle \textit{elem–type} \rangle ')')^{+}\ ')' \\ \langle \textit{attr–type} \rangle\ ::=\ [\textit{univalue}]\ |\ \textit{multivalue}, \end{array}$$

where *univalue* is an optional element, such that only multi-valued constants are mandatory to be specified. For example the template defining the class of *complex* numbers can be defined as $(complex\ (re\ real)\ (im\ real))$, while the template defining the class of lists having integer elements can be defied as follows:

$$(\textit{list}\ (\textit{elems multivalue integer})).$$

The template definition of all classes of facts represents the *first part* of the knowledge base of a rule-based expert system.

A *rule* represents a triplet specified by its unique names, its *antecedent* and its *consequent*:

$$\begin{array}{l} \langle rule \rangle\ ::=\ \langle name \rangle\ \textbf{if}\ \langle antecedent \rangle\ \textbf{then}\ \langle consequent \rangle\ \textbf{end} \\ \langle antecedent \rangle\ ::=\ \langle condition \rangle\ \{\textbf{and}\ \langle condition \rangle\}^{*} \\ \langle condition \rangle\ ::=\ \langle predicate \rangle\ '('\langle \textit{pattern} \rangle')' \\ \langle predicate \rangle\ ::=\ \textbf{Same}\ |\ \textbf{NotSame}\ |\ \textbf{LessThan}\ |\ \dots \\ \langle consequent \rangle\ ::=\ \langle action \rangle\ \{\textbf{and}\ \langle action \rangle\}^{*} \\ \langle action \rangle\ ::=\ \textbf{Add}'('\langle \textit{pattern} \rangle')'\ |\ \textbf{Del}'('\langle \textit{pattern} \rangle')', \end{array}$$

where the syntactic definition of ⟨*pattern*⟩ is similar to the definition of ⟨*fact*⟩, with the difference that constants can be replaced with variables:

$$\langle variable \rangle\ ::=\ ?\langle ident \rangle\ |\ !\langle ident \rangle,$$

The syntactic construction $?\langle ident \rangle$ represents a *uni-valued* variable, whose value can be a single constant, while $!\langle ident \rangle$ represents a *multi-valued* variable, whose value can be a sequence of constants from a give data type.

The predicate **Same** returns *true* if there is a matching between its associated pattern, and a fact from the fact base, while **NotSame** returns *true* if is not found any matching between its pattern and a fact from the fact base. Other predicates from the definition of a condition are usual logical predicates. Actions **Add** and **Del** add and delete facts to/from the fact base.

3.2 Translating the Template Facts

In the generic rule-based language presented in Sect. 5, facts are considered as a sequence of pairs (*attribute value*) having an associated name. When translating to Event-B, facts can be modelled by a tuple of constants, each constant belonging to a certain set, which is related to the corresponding attribute. For example, the fact ($complex\ re$ 7.1 im 5.3) can be translated to the sequence $\langle 7.1,\ 5.4\rangle$, which is a value of the cartesian product $complex = real \times real$.

A fact having only attributes with uni-valued values,

$$(ftype\ attr_1\ val_1\ attr_2\ val_2\ \ldots\ attr_n\ val_n), \tag{1}$$

having the template

$$(ftype\ (attr_1\ attr\text{–}type_1\ type_1)\ \ldots\ (attr_n\ attr\text{–}type_n\ type_n)), \tag{2}$$

can be translated to a tuple $\langle val_1,\ val_2,\ \ldots,\ val_n\rangle$, member of the set

$$type(ftype) = type_1 \times type_2 \times \cdots \times type_n. \tag{3}$$

Uni-valued constants are members of some appropriate sets, so Eqs. 1–3 are correct defined only for those facts having only attributes with uni-valued values. In the case of multi-valued constants we have to use sequences instead of sets. Unfortunately the mathematical language Event-B does not contain the definition of sequences. However sequences of values and their operators can be defined using the *Theory* plug-in [34]. We use the theory *Seq* from the standard library of the *Theory* plug-in.

Denoting by T a global data-type parameter, the *Seq* theory contains only polymorphic operators on type T. The main operator of the *Seq* theory is the *seq* operator, which has a parameter s representing a subset of T. *seq* defines the set of all sequences whose members are in s [34].

operator seq
  expression prefix
  arguments $s : \mathbb{P}(T)$
  direct definition
    $mset(f, n \cdot f \in (1..n) \rightarrow s \mid f\}$

In addition the *Seq* theory defines several operator and predicates, such as *seqSize*, *seqIsEmpty*, *seqHead*, *seqTail*, *seqAppend*, or *seqPrepend*, as well as several theorems and inference rules.

In order to translate the template facts from a KB to an Event-B model, we denote with **S** the family of basic sets representing the data types associated to the constants of all attributes from all template facts,

$$\mathbf{S} = \{S_1, S_2, \ldots, S_n\}. \tag{4}$$

Starting to **S** we can define two families of sets: the family, **U**, related to uni-valued attributes, and the family, **M**, related to multi-valued attributes. The family,

$$\mathbf{U} = \{S_{i_1}, S_{i_2}, \ldots, S_{i_u}\}, \tag{5}$$

is a subset of **S**, containing those ⟨*elem–type*⟩ for which:

$$\langle \mathit{attr–type}\rangle(\langle \mathit{fact–name}\rangle, \langle \mathit{attr–name}\rangle) = \mathit{univalue}.$$

In the case of multi-valued constants we use the *Seq* theory. Let **M** be the family related to multi-valued attributes, $\mathbf{M} = \{S_{j_1}, S_{j_2}, \ldots, S_{j_m}\}$, which is also a subset of **S** containing those ⟨*elem–type*⟩ for which:

$$\langle \mathit{attr–type}\rangle(\langle \mathit{fact–name}\rangle, \langle \mathit{attr–name}\rangle) = \mathit{multivalue}.$$

Based on **M**, let *Seq*(**M**) be the family of all sequences whose members are in the sets of the family **M**:

$$\mathit{Seq}(\mathbf{M}) = \{seq(S_{j_1}), \ldots, seq(S_{j_m})\}. \tag{6}$$

Now, the type *type*(*ftype*) as specified in Eq. 3 of the fact *ftype* specified in Eq. 1, can be generally defined as follows: for all $i \in \{1, \ldots, n\}$,

$$\begin{aligned} \mathit{elem–type}_i &= \mathit{univalue} \;\Rightarrow\; \mathit{type}_i = \mathit{attr–type}_i, \\ \mathit{elem–type}_i &= \mathit{multivalue} \;\Rightarrow\; \mathit{type}_i = seq(\mathit{attr–type}_i). \end{aligned}$$

Having the above family of sets and sequences constructed, we can construct a final family, **V**,

$$\mathbf{V} = \mathbf{S} \cup \mathit{Seq}(\mathbf{M}) = \{V_1, V_2, \ldots, V_p\}, \tag{7}$$

representing all possible data types associated to the attributes of all facts.

Based on the family **V**, we can now construct the data types associated to the template facts, which represents the *system of data types* of all possible facts:

$$\mathbf{D} = \mathrm{CP}(\mathbf{V}) = \{D_1, D_2, \ldots, D_k\}, \tag{8}$$

where, each set D_i is a cartesian product verifying the following condition:

$$\forall D_i \in \mathbf{D} \Rightarrow \exists i_1, i_2, \ldots, i_l \in \{1, 2, \ldots, p\} \cdot \; D_i = V_{i_1} \times V_{i_2} \times \cdots \times V_{i_l}. \tag{9}$$

We denote by **RD** the reunion of all these sets, $\mathbf{RD} = \bigcup_{1 \le i \le k} D_i$. In the translated model, the *fact base* of a rule-based system will be represented by a global variable that contains all current facts of the system. This constraint says that each fact from the fact base has a type defined in the *system of data types* **U**:

$$\forall f \in fb \Rightarrow \exists D \in \mathbf{RD} \cdot f \in D. \tag{10}$$

3.3 Translating the Rule Base

Each rule from a RB will be translated as a different event in the corresponding Event-B model. The event **INITIALIZATION** will add all initial facts of the ES in the fact base represented by the global variable fb, as defined in Eq. 10.

We can define a *first pattern* when translating a RB into an Event-B model. A condition **Same** of the form,

$$\textbf{Same}(\textit{ftype attr}_1\ ?x_1\ \textit{attr}_2\ ?x_2\ \ldots\ \textit{attr}_n\ ?x_n), \tag{11}$$

from a rule R, related to a fact template defined as in Eq. 2, can be translated to Event-B, in the any and $where$ parts of the associated event, as follows:

```
Event    $R \mathrel{\widehat{=}}$
    any
          $x_1, x_1, \ldots, x_n$
          …
    where
          $x_1 \in type_1 \wedge \ldots \wedge x_n \in type_n \wedge x_1 \mapsto x_2 \mapsto \ldots \mapsto x_n \in fb$
          …
    then
          …
    end
END
```

where, for each i, $type_i$ is defined as in Eq. 5.

There is no difference between uni-valued and multi-valued variables in the process of translation. The only restriction is that the data type of a multi-valued variable have to be a sequence of values.

The *second pattern* when translating a RB into an Event-B model refers to condition **NotSame**, which also launches the pattern matching process, but it has a different meaning,

$$\textbf{NotSame}(\textit{ftype attr}_1\ ?x_1\ \textit{attr}_2\ ?x_2\ \ldots\ \textit{attr}_n\ ?x_n), \tag{12}$$

related to a fact template defined also as in Eq. 2. The condition **NotSame**, can be the translated to an Event-B model, also in the any and $where$ parts of the associated event, as follow:

```
Event   R ≙
   any
         $x_1, x_1, \ldots, x_n$
         …
   where
         $x_1 \in type_1 \ \wedge \ \ldots \ \wedge \ x_n \in type_n \ \wedge \ x_1 \mapsto x_2 \mapsto \ldots \mapsto x_n \notin fb$
         …
   then
         …
   end
END
```

Because other conditions from the *antecedent* of a rule are only logical predicates, the *third pattern* says that for each logical predicate, an appropriate guard in the associated event can be written. For example, in the rule $AddSquare$, the condition **Equal**(?x ?y) has been translated as a simple guard, $x = y$.

Actions **Add** and **Del** can be translated into the $then$ part of the associated event. The *fourth pattern* refers to the **Add** action. Let us consider that a rule R contains the action:

$$\textbf{Add}(\textit{ftype attr}_1\ ?x_1\ \textit{attr}_2\ ?x_2\ \ldots\ \textit{attr}_n\ ?x_n),$$

related to a fact template defined also as in Eq. 2. The action **Add**, can be the translated to an Event-B model, in the $then$ part of the associated event, as follow:

```
Event   R ≙
   any
         $x_1, x_1, \ldots, x_n$
         $y_1, y_1, \ldots, y_m$
         …
   where
         …
   then
         $fb' = fb \cup \{x_1 \mapsto x_2 \mapsto \ldots \mapsto x_n\}$
         …
   end
END
```

The *fifth pattern* refers to the **Del** action, which is similar to the previous pattern. The only difference consists in the set operation performed on the fb variable. The action

$$\textbf{Del}(\textit{ftype attr}_1\ ?y_1\ \textit{attr}_2\ ?y_2\ \ldots\ \textit{attr}_n\ ?y_n),$$

can be translated into a $then$ part of the associated event as follows:

```
Event   R ≙
   any
         $x_1, x_1, \ldots, x_n$
         $y_1, y_1, \ldots, y_m$
         …
```

```
  where
      …
  then
      $fb' = fb \setminus \{y_1 \mapsto y_2 \mapsto \ldots \mapsto y_m\}$
      …
  end
END
```

Remark Because **Add** and **Del** actions perform set operations on the same global variable, fb, these actions can be combined into a single action.

3.4 A Modelling Example

As an example we translate a simple *CLIPS* program to an Event-B model. The following CLIPS program determines if an initial string is palindrome.

```
(deffacts facts (string symbols 1 0 1))

(defrule empty-string
   ?c <- (string symbols)
   =>
   (retract ?c)
   (printout t "The initial string is palindrome" crlf))

(defrule singleton-string
   ?c <- (string symbols ?)
   =>
   (retract ?c)
   (printout t "The initial string is palindrome" crlf))

(defrule equal-symbols
   ?c <- (string symbols ?symbol $?middle ?symbol)
   =>
   (retract ?c)
   (assert (string symbols $?middle)))

(defrule different-symbols
   ?c <- (string symbols ?symbol1 $? ?symbol2)
   (test (<> ?symbol1 ?symbol2))
   =>
   (retract ?c)
   (printout t "The initial string is not palindrome" crlf))
```

The first step is to rewrite the above RB in the language presented in the Sect. 3.1. There are only two finite sets used in this knowledge:

$$symb = \{0, 1\},\ res = \{OK, NOK, NULL\}.$$

We have two fact patterns,

$(string\ (symbols\ multivalue\ symb))$
$(result\ (type\ res))$

and the following rule base:

```
empty : If Same(string symbols)
then Del(string symbols) and Add(result type OK) End

singleton : If Same(string symbols ?s)
then Add(result type OK) End

equal : If Same(string symbols ?s !middle ?s)
then Del(string symbols ?s !middle ?s)
    and Add(string symbols !middle) End

different : If Same(string symbols ?s1 !middle ?s2)
    and NotEqual(?s1 ?s2)
then Del(string symbols ?s1 !middle ?s2)
    and Add(result type NOK) End
```

According to the Sect. 3.2, the following sets are constructed:

$$\mathbf{S} = \{symb, res\},\ \mathbf{U} = \{res\},\ \mathbf{M} = \{symb\},\ \mathbf{V} = \{res, Seq(symb)\}.$$

The translated Event-B model of the previous knowledge base contains a context,

```
CONTEXT     Palindrom_C
CONSTANTS

        0, 1
        OK, NOK, NULL

SETS

        SYMB, RES

AXIOMS

        partition(SYMB, {0}, {1})
        partition(RES, { OK }, { NOK }, { NULL })

END
```

and a machine,

```
MACHINE     Palindrom_M
```

```
SEES    Palindrom_C
VARIABLES

        $fb\ sq\ r$

INVARIANTS

        $fb \in RES \ \wedge \ fb \in seq(SYMB) \ \wedge \ sq \in seq(SYMB) \ \wedge \ r \in RES$

EVENTS
Initialisation

    begin
            $fb := \emptyset$
            sq := empySeq
            sq := seqAppend(seqAppend(seqAppend(sq, 1), 0), 1)
            r := $NULL$
            $fb := fb \cup \{sq\}$
    end

Event   $empty \mathrel{\widehat{=}}$

    when
            $sq \in fb \ \wedge \ seqIsEmpty(sq)$
    then
            r := $OK$
            $fb := fb \cup \{r\} \setminus \{sq\}$
    end

Event   $singleton \mathrel{\widehat{=}}$

    when
            $(sq \in fb) \ \wedge \ (seqSize(sq) = 1)$
    then
            r := $OK$
            $fb := fb \cup \{r\} \setminus \{sq\}$
    end

Event   $equal \mathrel{\widehat{=}}$

    any
            s1, middle, s2
    where
            $s1 \in SYMB \ \wedge \ s2 \in SYMB \ \wedge \ s1 = s2 \ \wedge \ middle \in seq(SYMB)$
            sq = seqAppend(seqPrepend(middle, s1), s2)
    then
            $fb := fb \cup \{middle\} \setminus \{sq\}$
    end

Event   $different \mathrel{\widehat{=}}$

    any
            s1, middle, s1
    where
            $s1 \in SYMB \ \wedge \ s2 \in SYMB \ \wedge \ s1 \neq s2 \ \wedge \ middle \in seq(SYMB)$
            sq = seqAppend(seqPrepend(middle, s1), s2)
    then
            r := $NOK$
            $fb := fb \cup \{r\} \setminus \{sq\}$
    end

END
```

4 Verifying the Knowledge Base of an Rule-Based System

We study the verification of a RB of an ES by using the translated Event-B model, as presented in Sect. 3. Two steps are performed when verifying a knowledge base: (a) checking the *model consistency* of the translated knowledge base, by using the *ProB* plugin, and (b) checking the main *properties related to the verification task*, by using both the *ProB* plug-in, and the mathematical language of the *Event-B* language.

Model checking is a term from the SE domain [37], which represents an automatic approach to formal verification of a model based on state exploration. In the case of Event-B and its associated tool, Rodin, there are two main model checking activities: *consistency checking*, which allows to verify if the operations of a machine preserve its invariant, and *refinement checking*, which allows to verify if a concrete machine is a valid refinement of its related abstract machine. Because we do not use the refinement action in the translated Event-B model, in our case we will use only *consistency checking* of an Event-B model.

Model consistency is checked in Event-B and Rodin by verifying two conditions: a *feasibility* condition, and an *invariant preservation* condition. Let M be a machine having the invariant $I(v)$, v its sequence of variables, and an event, ev_m defined as:

$ev_m \mathrel{\widehat{=}}$ any p_m where $G_m(v)$ then $S_m(v, v')$ end.

The two conditions verified in a model checking action can be described as follows:

- The *feasibility* condition, which ensures that s_m provides an after state whenever G_m holds,
- The *invariant preservation* condition, which allows invariants to hold whenever variable values change.

Similar conditions are necessary to the initialisation event.

The first step of the RB verification, model consistency verification, can be automated performed by using the ProB plug-in on the translated Event-B model. For the example presented in Sect. 3.2, all these proof obligations for machine consistency are generated and discarded automatically by the *ProB* plug-in. So, for the model-checking point of view, the example presented in Sect. 3.2 is consistent.

The second step in the checking for consistency of a RB task performs the verification of some important properties, such as [38]:

1. *Redundant rules*. Two rules are redundant if they succeed having the same condition and the same conclusion.
2. *Subsumed rules*. One rule is subsumed by another if the two rules have the same conclusions, but one of them contains additional conditions.
3. *Conflicting rules*. Two rules are in conflict if they succeed in the same condition but have conflicting conclusions.
4. *Circular rules*. A set of rules is circular if the chaining of these rules form a cycle.

These properties are called *static properties* because the research in the domain of RB verification has been oriented only on rule-based systems that not use variables, and in this case the above properties check the structure and interconnection of the information in the RB. In the case of rule-based systems that use variables not all above properties can be statically checked, because the basic element of a RB that can fire is not a rule, but a rule instance. As a consequence, verifying the structure of a RB is not enough for some properties. In fact, a sound theory for checking consistency of a RB that uses variables is not developed until now.

In the following we will adapt the above four properties for rule-based systems that use variables and forward inference chaining. The first two properties can be also statically and syntactically checked in our approach, because they verify structural properties. Let ev_i and ev_j the events translated from the rules r_i and r_j. In this case:

(a) The rules r_i and r_j are *redundant* if $G_i(v) = G_j(v)$ and $S_i(v, v') = S_j(v, v')$,
(b) The rule r_j is *subsumed* by r_i if $G_i(v) \Rightarrow G_j(v)$ and $S_i(v, v') = S_j(v, v')$.

The third property represents also a static property, but it can not always be syntactically verified. With the above notations we can formalize this property as follows:

(c) The rules r_i and r_j are in *conflict* if $G_i(v) = G_j(v)$ and *Conflict*$(S_i(v, v'), S_j(v, v'))$,

where *Conflict*$(S_i(v, v'), S_j(v, v'))$ means that some actions in $S_i(v, v')$ and $S_j(v, v')$ are in conflict.

Because there are only two possible actions, **Add**(f) and **Del**(f), there can be two types of conflicts. Denoting by fb the variable of the translated Event-B model representing the fact-base, by ev_i and ev_j two events, and by f_1 and f_2 respectively, the Event-B variables representing the facts (for simplicity, with the same name) f_1 and f_2, the possible conflicts are: (i) there are at least two opposite actions in S_i and S_j, that is adding and deleting the same fact, or (ii) there are at least two actions in S_i and S_j that add two facts, f_1 and f_2, that are incompatible.

The first conflict can be also syntactically checked, and it can be formalized as follows:

(c1) There exists $fb' = bf \cup \{f\} \in S_i$, and $fb' = bf \setminus \{f\} \in S_j$, where f is the same fact in the two actions.

Unfortunately the second conflict cannot be formalized due to lack of semantic component of the description language presented in Sect. 2.1. For example, the following facts, $(father\text{–}of\ father\ John\ son\ Jim)$ and $(father\text{–}of\ father\ Jim\ son\ Jhon)$ may be in conflict if their meaning say that "John is the father of Jim" and "Jim is the father of John" respectively. Because the generic language presented in Sect. 2.1 is inspired from the CLIPS and Jess languages, where facts are ordered sequences of constants, such type of conflict cannot be detected. Except this conflict, the above three properties can be statically verified for the example presented in Sect. 3.2.

The last property (circular rules) cannot be statically checked for rule-based expert systems that use variables. For modeling the dynamic evolution of a system modeled by Event-B language we need to represent the sequence of successive states and their corresponding events. We use the notion of *event traces*, which can model the system behaviour, as in well-known from process algebra, especially CSP. Although event traces are not part of the standard semantic definitions in Event-B, the ProB animator plug-in can be viewed as computing possible traces of an Event-B machine [33].

ProB uses the notion of *rechablable states*. The first state, v_{init} results after the initialisation event, $init$. If in a state v, the guard $G_i(v)$ of an event e_i is true, and if after performing its related actions $S_i(v, v')$, the new state of the model is v': $v \overset{e_i}{\to} v'$. If in this new state, another event, e_j has a true guard, $G_j(v')$, the system can reach a new state, v'', $v \xrightarrow{e_i} v' \xrightarrow{e_j} v''$, such that $S_j(v', v'')$ is true. In this case we have both a *trace of events* $\langle e_i, e_j \rangle$, and a *trace of states* $\langle v, v', v'' \rangle$.

Thus, a *trace of events* is a sequence $\langle e_{i_0}, e_{i_1}, e_{i_2}, \ldots, e_{i_n}, \ldots \rangle$ that correspond to a *trace of states* $\langle v_{i_1}, v_{i_2}, \ldots, v_{i_n}, v_{i_{n+1}}, \ldots \rangle$, where $e_{i_0} = init$, and for each $j > 1$, $v_{i_{j-1}} \xrightarrow{e_{i_j}} v_{i_j} \xrightarrow{e_{i_{j+1}}} v_{i_{j+1}}$. The two types of traces are not independent. Having a reachable state, v, and a sub-trace of events $te = \langle e_{k_1}, e_{k_2}, \ldots, e_{k_m} \rangle$, the corresponding sub-trace of states is $ts = Im(v, te) = \langle v_{k_2}, v_{k_3}, \ldots, v_{k_{m+1}} \rangle$, such that:

$$\begin{aligned} &v \xrightarrow{e_{k_1}} v_{k_2}, \\ &v_{k_j} \xrightarrow{e_{k_{j+1}}} v_{k_{j+1}}, \forall j \in \{1, 2, \ldots, m\}. \end{aligned} \tag{13}$$

Now we can formalize the condition of *circular rules*. Let $t = s v_{i_1} v_{i_2} \ldots v_{i_k} u$ a trace of states, where s and u represent sequences of states, $t' = \langle e_{i_2}, e_{i_3}, \ldots, e_{i_k} \rangle$ a sub-trace of events, such that $t' = Im(v_{i_1}, \langle v_{i_2}, v_{i_3}, \ldots, v_{i_{k+1}} \rangle)$. The rule base contains *circular rules* if the following condition hold:

$$(v_{i_1} = v_{i_{k-1}}) \wedge (e_{i_2} = e_{i_k}), \tag{14}$$

Circular rules can be detected in the ProB animator, when the same event appear twice in the same state. For the example presented in Sect. 3.2 there are not circular rules.

5 Conclusion

The research of this paper is related to the study of the consistency in knowledge base of a rule-based expert system that uses variables and forward chaining inference. We propose to use the Event-B method to model the knowledge base and its associated Rodin platform. We used two plug-ins of *Rodin*: (a) the Theory plug-in for the construction of data types needed to translate the knowledge base to an Event-B model and (b) the ProB plug-in for model checking and for checking the consistency of the translated Event-B model. We use some SE methods in order to perform specific KBE tasks, related to verification of KBs.

Our approach is different from other proposals because (a) it allows to verify rule bases containing variables, and (b) it uses a formal method, Event-B, and its supporting tool, Rodin, in order to perform automatically the most part of the verification process.

Our future research is twofold: (a) to extend this model to include an inference engine for a forward chaining algorithm, allowing to simulate a rule-based system before its implementation phase, and (b) to extend the research to verify the consistency of the fact base of an rule-based system.

References

1. Adrion, W., Branstad, M., Cherniavsky, J.: Validation, verification and testing of computer software. Comput. Surv. **14**(2), 159–192 (1982)
2. Meseguer, P.: Towards a conceptual framework for expert system validation. AI Commun. **5**(3), 119–135 (1992)
3. Laurent, J.P.H.: Proposals for a valid terminology in KBS validation. In: Proceedings of the European Conference on Artificial Intelligence - ECAI, pp. 829–834. Wiley, Vienna, Austria (1992)
4. Hoppe, T., Meseguer, P.: On the terminology of VVT: a proposal. IEEE Expert **93**, 48–55 (1993)
5. Sommerville, I.: Software Engineering. Addison-Wesley, USA (2011)
6. Meseguer, P., Preece, A.: Verification and validation of knowledge-based systems with formal specifications. Knowl. Eng. Rev. **10**(4), 331–343 (1995)
7. Preece, A.: Evaluating verification and validation methods in knowledge engineering. Industrial Knowledge Management, pp. 91–104. Springer, London (2001)
8. O'Keefe, R.M., O'Leary, D.E.: Expert system verification and validation: a survey and tutorial. Artif. Intell. Rev. **7**(1), 3–42 (1993)
9. Meseguer, P.: A new method to checking rule bases for inconsistency: a petri net approach. In: Proceedings of ECAI 90, pp. 437–442. Espoo, Finland (1990)
10. He, X., Yang, W.C.H., Yang, S.: A new approach to verify rule-based systems using petri nets. In: Proceedings of 23th Annual International Computer Software and Applications Conference, pp. 462–467. Los Alamitos, CA, USA (1999)
11. Wu, C.H., Lee, S.J.: Enhanced high-level petri nets with multiple colors for knowledge verification/validation of rule-based expert systems. IEEE Trans. Syst. Man Cybern. Part B: Cybern. **27**(5), 760–773 (1997)
12. Ramaswamy, M., Sarkar, S., Sho, C.Y.: Using directed hypergraphs to verify rule-based expert systems. IEEE Trans. Knowl. Data Eng. **9**(2), 221–237 (1997)
13. Gursaran, G., Kanungo, S., Sinha, A.: Rule-base content verification using a digraph-based modelling approach. Artif. Intell. Eng. **13**(3), 321–336 (1999)
14. Laita, L., Roanes-Lozano, E., de Ledesma, L., Alonso, J.: A computer algebra approach to verification and deduction in many valued knowledge systems. Soft Comput. **3**(1), 7–19 (1999)
15. Pierret-Golbreich, C., Talon, X.: TFL: an algebraic language to specify the dynamic behaviour of knowledge-based systems. Knowl. Eng. Rev. **11**, 253–280 (1996)
16. Baumeister, J., Seipel, D.: Anomalies in ontologies with rules. Web Semant. Sci. Serv. Agents World Wide Web **8**(1), 55–68 (2010)
17. Krotzsch, M., Rudolph, S., Hitzler, P.: ELP: tractable rules for OWL. In: Proceedings of the 7th International Semantic Web Conference - ISVC, pp. 649–664. Springer (2008)
18. Event-B, the Rodin Platform. http://www.event-b.org/
19. ProB. http://www.stups.uni-duesseldorf.de/ProB/index.php5/ProB_for_Rodin

20. Woodcock, J., Davies, J.: Using Z: Specification, Refinement and Proof. Prentice Hall, London (1996)
21. Smith, G.: The Object-Z Specification Language. Springer, US (2000)
22. Jones, C.: Systematic Software Development Using VDM. Prentice-Hall International, Englewood Cliffs (1986)
23. Abrial, J.R.: The B Book: Assigning Programs to Meanings. Cambridge University Press, Cambridge (1996)
24. Abrial, J.R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press, Cambridge (2010)
25. Kropf, T.: Introduction to Formal Hardware Verification. Springer, Berlin (1998)
26. Lamport, L.: The temporal logic of actions. ACM Trans. Program. Lang. Syst. **16**(3), 872–923 (1994)
27. Hoare, C.: Communicating Sequential Processes. Prentice-Hall International, Englewood Cliffs (1985)
28. Milner, R.: Communication and Concurrency. Prentice-Hall International, New York (1989)
29. Bidoit, M., Mosses, P.: CASH User Manual: Introduction to Using the Common Algebraic Specification. Lecture Notes in Computer Science, vol. 2900. Springer, Berlin (2004)
30. Goguen, J., Malcolm, G.: Algebraic Semantics of Imperative Programs. MIT Press, Cambridge (1996)
31. Deploy. http://www.deploy-project.eu/
32. Advance. http://www.advance-ict.eu/
33. Leuschel, M., Butler, M.: ProB: an automated analysis toolset for the B method. Int. J. Softw. Tools Technol. Transf. **10**(2), 185–203 (2008)
34. Butler, M., Maamria, I.: Practical theory extension in event-B. Theories of Programming and Formal Methods. Lecture Notes in Computer Science, vol. 8051, pp. 67–81. Springer, Berlin (2013)
35. CLIPS. http://clipsrules.sourceforge.net/
36. Jess. http://herzberg.ca.sandia.gov/jess/
37. Chan, W., Anderson, R.J., Beame, P., Burns, S., Modugno, F., Notkin, D., Reese, J.D.: Model checking large software specifications. IEEE Trans. Softw. Eng. **24**(7), 498–520 (1998)
38. Nguyen, T., Perkins, W., Pecora, D.: Knowledge base verification. AI Mag. **8**(2), 69–75 (1987)

Knowledge Engineering for Distributed Case-Based Reasoning Systems

Kerstin Bach

Abstract This chapter describes how to identify and collect human knowledge and transform it into machine readable and actionable knowledge. We will focus on the knowledge acquisition for distributed case-based reasoning systems. Case-based reasoning (CBR) is a well-known methodology for implementing knowledge-intensive decision support systems (Aamodt, Plaza, Artif Intell Commun, 7(1):39–59, 1994) [1] and has been applied in a broad range of applications. It captures experiences in the form of problem and solution pairs, which are recalled when similar problems reoccur. In order to create a CBR system the initial knowledge has to be identified and captured. In this chapter, we will summarise the knowledge acquisition method presented by Bach, Knowledge acquisition for case-based reasoning systems. Ph.D. thesis, University of Hildesheim, München (2012) [2] and give an running example within the travel medicine domain utilising the open source tool for developing CBR systems, myCBR.

Keywords Case-based reasoning · Knowledge acquisition · Knowledge-based systems · Knowledge engineering · Distributed knowledge acquisition

1 Introduction and Motivation

Following our approach of Collaborative Multi-Expert Systems [3] the knowledge sources, which are used to store and provide knowledge, are mostly distributed. When dealing with complex application domains it is easier to maintain a number of heterogeneous knowledge sources than one monolithic knowledge source. Therefore we propose a Knowledge Line that holds a map of topics, which split rather complex knowledge in smaller, reusable units (knowledge sources). Moreover, the

K. Bach (✉)
Department of Computer and Information Science, Norwegian University of Science and Technology, Trondheim, Norway
e-mail: kerstin.bach@ntnu.no
URL: http://www.idi.ntnu.no

G.J. Nalepa and J. Baumeister (eds.), *Synergies Between Knowledge Engineering and Software Engineering*, Advances in Intelligent Systems and Computing 626, https://doi.org/10.1007/978-3-319-64161-4_7

knowledge sources contain different kinds of information as well as there can also be multiple knowledge sources for the same purpose. Therefore each source has to be described in order to be integrated in a retrieval process which uses a various number of knowledge sources. In the following we describe how multiple CBR system holding case of different topics, but aggregate their results eventually, can be built. This knowledge acquisition method is targeted for capturing heterogeneous knowledge sources and build a distributed CBR system that maintains its knowledge in homogeneous containers that are queries on demand. Therewith it fits in the requirements of knowledge acquisition for the SEASALT architecture [4] as well as the CoMES approach [3].

The goal of the knowledge modularization is building as independent modules as possible in order to reduce the complexity of each individual CBR system. Modules can be described as task specific program parts as they are described in [5]. The knowledge modularization in SEASALT aims are minimizing dependencies between Topic Agents by identifying modules that are coherent within themselves. Modules, further on, can be combined as required within a Knowledge Line. The result of a problem solving system which is based on SEASALT is always a solution that consists of partial solutions, which originate in heterogeneous partial domains.

For generating a solution within a Knowledge Line a Knowledge Engineer must define the overall contexts for ensure the composition of an overall result. For that reason the Knowledge Modularization is a basis of the Knowledge Composition.

Crucial for the effective application of such knowledge intensive systems is the organization of knowledge. For that reason the conceptional (development) phase requires special attention, because after that phase the development of the CBR systems can be carried out individually. The Knowledge Line contains on the one hand the Knowledge Map for organizing information about each module (Topic Agent) that is required for combining the snippets, which are the partial solutions of which the final result consists of.

2 Background and Related Work

Two related methodologies to the work presented here are DISER [6] and INRECA [7]. While both, DISER and INRECA are methodologies for creating single CBR systems, the process of building a knowledge line instead describes the systematic development of decentralized, CBR systems.

DISER describes a methodology for developing experience-based systems in general. It especially focuses on the integration of the CBR system in an enterprise rather than providing information from Web 2.0 sources to laymen. Furthermore, the knowledge line requires knowledge engineers to execute key tasks while in [6] enterprise executives are addressed with the goal to integrate the system in existing socio-technical processes. With SEASALT instantiations, we are focusing more on the technical realization rather than the social interaction between stakeholders. When recalling DISER's development aspects, then a Knowledge Line can be positioned in the vision to pilot phases. The presented methodologies addresses also maintenance aspects, but has not a particular phase, because novel information is constantly fed into the systems due to the stream-like data in web forums.

INRECA on the other hand was particularly focused on developing CBR systems and provides experiences for the development itself on various abstraction levels. Because of the specialization to CBR, common experiences derived from the development of various CBR systems, can be generalized and shared. INRECA does not cover any experiences for developing distributed systems such as the previously described snippet descriptions. Eventually a Knowledge Line can be seen as an addition to the INRECA recipes since it presents an approach for distributed CBR systems, which is has not been covered before.

On INRECAs *Common Generic Level* contains more abstract processes for the development of any kind of CBR system. The herewith introduced methodology focuses on a subset of systems that are based on heterogeneous CBR systems. For that reason, only the abstract process presented in Fig. 3 would fit that generic level. It can for instance be applied for distinguishing whether a distributed system is required to fit the expectations or not. The *Specific Project Level* is not directly addressed with the work previously present, but the experiences made during the instantiation of an application such as docQuery could provide extensions to that level.

3 Knowledge Modularization

In this section we are introducing a process model for the knowledge modularization within a Knowledge Line. This process model, describes the goal-oriented development of decentralized, heterogeneous case bases and supports the Knowledge Engineer with a structured approach for modularizing knowledge based on available data. The approach presented in this work does not aim at distributing knowledge for performance reasons, instead we specifically extract information for the respective knowledge sources from Internet communities or to have experts or Knowledge Engineers maintaining one knowledge base. Hence, we are creating knowledge sources, especially Case-Based Reasoning systems, that are accessed dynamically according to the utility and accessibility to answer a given question. Each retrieval result of a query is a part of the combined information as it is described in the CoMES approach. The representation of the knowledge modularization is the Knowledge Map, which will be described in the following subsection. The successive subsection will introduce the Knowledge Line process model and describe the containing sub-processes.

3.1 Knowledge Map

The **Knowledge Map** organizes all available knowledge sources that can be accessed by a Coordination Agent that creates individual requests and combines information.

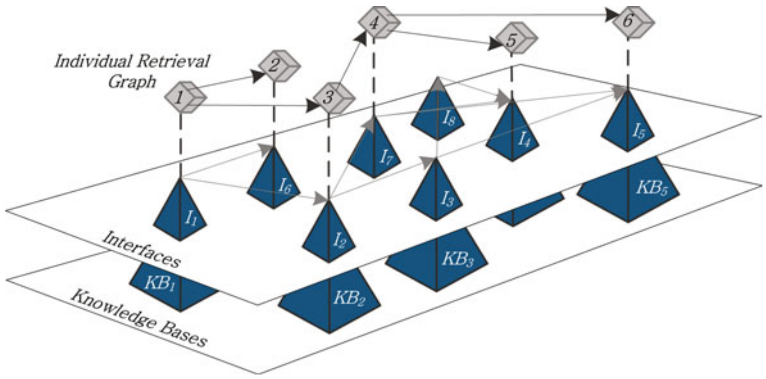


Fig. 1 Knowledge Map containing topic agents and knowledge bases

The term Knowledge Map has been introduced by [8] describing working knowledge in an organization. In the aforementioned book they describe a knowledge map from the organizational point of view in which human experts are mapped to topics or expertise fields in order to ensure that everybody in a company knows who is an expert in a certain domain. We transfer this concept in an intelligent agent framework that coordinates different knowledge sources.

A Knowledge Map (KM) consists of a number of Topic Agents TA that are depending on each other and each consist of a software agent A on top of a knowledge base KB. Thus it can be defined as follows:

$$KM = \{TA_1, TA_2, TA_3, \ldots, TA_n\} \ with \ \ TA = (KB, A) \tag{1}$$

A Topic Agent is a knowledge-based system itself and the software agent queries it. The Topic Agent collaborates with the Coordination Agent that navigates through the Knowledge Map and asks subsequent questions to the individual Topic agents thus creating an individual path through the map. There are dependencies $Dep_{constraint}$ between the Topic Agents which define that sequence and influence the retrieval executed by one of the subsequent Topic Agents. A dependency exists if one agent's output serves as another agent's input and thus enforces a subsequent query. Since the dependencies between Topic Agents can take any form, we decided to implement the Knowledge Map as a graph where each Topic Agent is represented by a node and directed edges denote the dependencies.

Figure 1 shows a Knowledge Map containing the knowledge bases and software agents as well as an example for a possible path through the knowledge sources.

3.2 *Pre-processing*

Based on a customers requirement for a target system, the Knowledge Engineer defines the systems functionalities as well we the overall goal. We assume that the overall goal of the knowledge-based system is the composition of results retrieved from multiple, heterogeneous case bases. Examples for this are the menu creation in the cooking domain [9, 10] or the travel medicine domain as it will be used as an example further on.

Within docQuery, we require to use information of various CBR systems such as regional information, information on diseases, information on vaccinations and medications or activities. There are dependencies between each case base, but each one covers its own domain: For instance the case base disease contains information about infectious diseases that work outside of docquery, i.e. in a medical information system, as well. The dependencies or links between case bases are necessary to compile first a retrieval graph over the knowledge map and eventually compose the solution integrating all retrieved cases.

3.2.1 Knowledge Source Sounding

After identifying the scope of each individual case base, existing and available knowledge sources have to be investigated whether they can provide the required data and quality. The major goal of this first step is getting an overview of

- What kind of knowledge already exists?
- What has to be reviewed?
- What has to be acquired in order to fit the given requirements?

Knowledge sources are mainly data collections, but also cover experts for certain domains.

For the docQuery use case we looked at what type of knowledge is naturally available. This led us to the CBR systems Region, Disease, Medicament, Hospital and Activity. It is obvious that these information can be plugged in any other application as well. For example, the Region case base can provide information in a travel agency scenario. Based on the discussions with domain experts we further identified the need of the case bases Chronic Illnesses and Associated Condition. Chronic Illnesses and Disease are separated because of their handling in a knowledge map and the information provided about them. While chronic illnesses are information to be specified by the user, docQuery provides prevention of diseases. Further, explanations on chronic illnesses are focused on how to travel with that handicap, while the disease are potentially unknown and we try to avoid the user with an infection. Therefore disease and vectors have to be explained, while chronic illnesses are known to that particular person. The associated condition case base mainly contains prevention information for any kind of occasion. These information can be linked to most of the other case bases and therefore this case base has to be queried towards the end of the querying process.

This example shows that the identification is at least a 2-step-process in order to define the scope of each CBR system. However, for defining attributes and links usually more iterations are necessary. These discussion are led by the Knowledge Engineer, which increases his/her awareness of the domain. Common sources that should be initially considered and were used within the development of docQuery are DBpedia[1] or Google Knowledge Graph.[2]

3.2.2 Identification and Representation of Snippets

The modularization of the required knowledge is carried out based on the knowledge engineer's understanding of the domain aiming at fitting the given requirements. The main task is the identification and definition of homogeneous and independent modules that keep their semantic even when they are seen as singletons (*independence*). Each module has to be combinable in order to complement another module (*compatibility*) and requires rules or constraints ensuring a valid combination of the modules (*validity*). The requirements *independence* and *compatibility* are focusing on the information sources while *validity* aims at procedural knowledge applied combining the results.

Within SEASALT the previously described modules are called snippets since the concept is similar to [11]. After their identification, the knowledge sources providing data for the snippet cases have to be determined. Mainly the Knowledge Engineer has to decide which information from sources are included. For the composition of information that prevent travelers from diseases. Since our modularization happens with respect to different sub-domains, the individual case bases that are the result of this modularization are not absolutely independent of each other. Instead they have a net of dependencies between them that indicates what other case bases are affected by changes in one individual case base. These dependencies also affect some of the aforementioned maintenance tasks and split them into two different kinds: those which have to take other case bases into consideration and those which do not. A case factory agent that performs the task of maintaining a case base's uniqueness, minimality, incoherence or consistency can do this without knowing about other case bases. An agent that inserts new cases or adds new data to existing cases is a different matter. For instance if a new case is inserted into the diseases case base, or an existing disease breaks out in a new region, the inserting agent has to check, whether every risk region indicated in the respective disease's regions attribute is actually included in the country case base's underlying knowledge model, otherwise this diseases case will never be retrieved. Another example of such dependencies would be, if one of the regions can not be associated, a domain expert will have to be contacted and asked for specifications, since we assume that in this case there is either a simple typing error, a synonymous name for a region, or the new region will have to be added to the ontology by a domain expert. The same is true for new data being added to the medicament case base. Here the area of application has to yield at least one result in the disease case base, otherwise the new data have again to be passed to a domain expert for a review. Although this approach is rather maintenance-intensive, our medical application scenario requires this very conservative case addition strategy in

[1]http://dbpedia.org/.

[2]https://developers.google.com/knowledge-graph/.

order to preserve the system's overall accuracy and the case factory approach allows us to realize this strategy and also adapt to new dependencies, should they arise [3].

As an example, travel medical data contain information about countries, diseases, medications, vaccinations as well as descriptions, guidelines, and experiences. Therefore the knowledge in docQuery will be provided in case bases and each case base will contain one specific topic with its own domain model and maintenance agents/jobs. However, following the modularized structure of knowledge in docQuery, CBR agents will be used for each individual topic agent providing information. Aiming at higher accuracy each case base will serve its own topic and the case format will exactly fit for the type of knowledge. Furthermore the case bases will contain similarity measures, adaptation knowledge and the vocabulary to represent and retrieve cases. In travel medicine it can be dangerous to use CBR for the whole set of information, because the combination of medications, vaccinations, side effects, contraindications, etc. regarding the traveler's health history have to be correct, without any contradicting information. Instead of that we will apply CBR for each topic and do the combination of the responses afterwards using the constraints given in the response sets. Each issue handled in a case base will be provided using CBR methods and the strength of CBR, finding similar information on a given topic, will ensure a higher quality of information provision.

Now we will exemplify four docQuery case bases that are each representing one topic and explain the dependencies between them. The selected case bases are examples to explain our approach and for the implementation of docQuery there will be at least six more case bases. The case base country will contain specific country information a traveler has to consider planning a journey. Furthermore the information will be separated in the sections a traveler has to pay attention to before, during, and after the journey. The country information also includes the required vaccinations and additional information, for example guidelines for a healthy journey. The case base disease holds more than 100 diseases considered in a travel medical consultation. It concentrates on diseases that might affect a traveler on a journey, for instance Malaria, Avian Influenza, or Dengue. A disease in this case base is characterized by general information on the disease, how to avoid the disease, how to behave if one has had the disease before, and how to protect oneself. The third case base we will introduce is medicament with details about medicaments and their area of application (diseases, vaccinations, age, etc.). Basically it contains information about active pharmaceutical ingredients, effectiveness, therapeutic field, contraindication, inter-dependencies, and the countries in which those medicaments are approved. In diseases we do not store information on chronic illnesses, be-cause they will be modeled in their own case base. Instead we focus on diseases which can affect travelers during their journey. The fourth case base will hold activities which are used within docQuery to provide safety advice for intended activities when planning a journey. For travelers, activities are the major part of their journey, but may involve certain risks for which safety advice is needed and furthermore while asking for their plans they usually describe their activities which we can use to provide better guidance. Examples of such activities are diving, hill-climbing or even swimming.

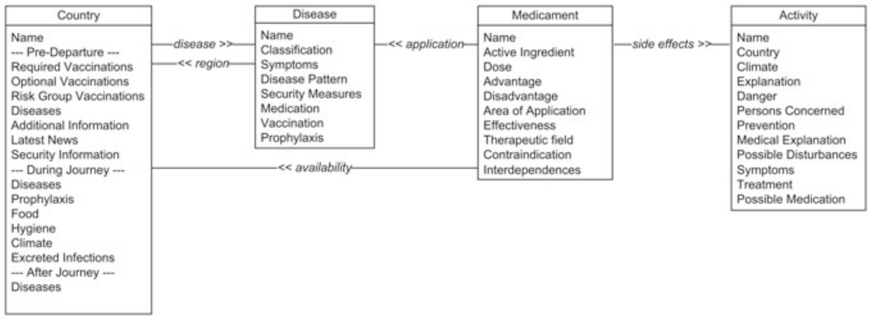


Fig. 2 Case representations for four case bases that provide knowledge for the topic agents

Complete travel medical information will contain knowledge of all four case bases enhanced with descriptions, guidelines, and previous experiences. The combination of the information retrieved from each case base will be done by a Coordination Agent as it can be seen in the SEASALT architecture (Fig. 2). The coordination agent will request each agent and based on the agents' response and the given information by the traveler the next request containing all constraints to another topic agent will be created and send.

4 Knowledge Modularization Methodology

In Sect. 3 we described how to implement the knowledge modularization for the travel medicine domain. In this showcase we explained how to proceed in order to develop CBR systems which will be deployed in the SEASALT architecture. In this section we will discuss a general methodology how to modularize a domain for setting up a knowledge line.

For the development of CBR systems within SEASALT we use snippet descriptions[3] for describing pivotal factors (or knowledge) as topic agents. Like in traditional CBR systems each snippet description requires its own representation including their knowledge containers. With regard to decentralization and distribution of the content (i.e. the snippets) the Knowledge Engineer has to define the topics, the CBR system's specifications and the linkage between topics. Further on it has to be ensured that a composition of several snippets, which are still meaningful, can be achieved.

The goal of this section is not to describe how to design a CBR system, it presents a methodology for the development of distributed CBR systems, which are based on the SEASALT architecture. In contrast to INRECA and DISER we are not aiming at

[3]In the remaining parts of this work we will differentiate between *snippet descriptions* and *snippet*. Snippet descriptions are the conceptual representation of knowledge and are equivalent to a case representation. Snippets on the other hand are instances of snippet descriptions and therewith equivalent to cases.

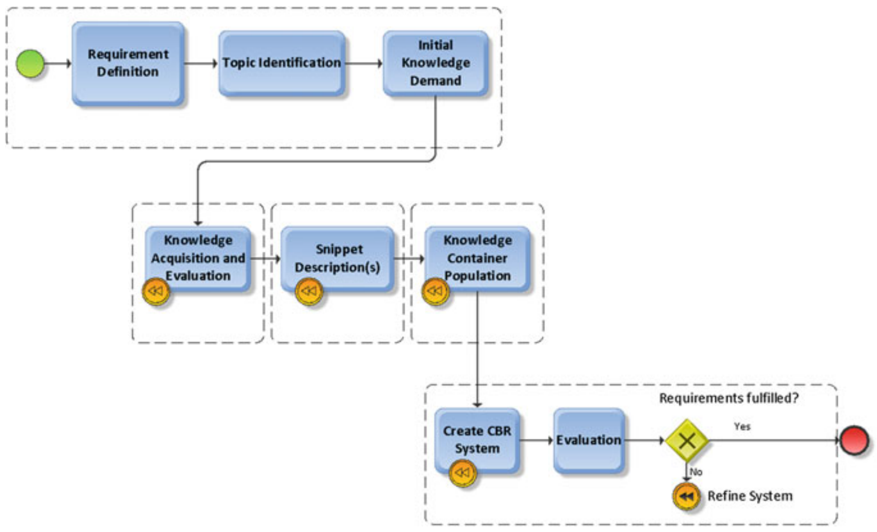


Fig. 3 Abstract knowledge modularization process

providing a complete methodology that can directly be implemented in an enterprise. More precisely we show how to analyze an application domain and identify topics and snippet descriptions.

The overall process can be segmented in subprocesses, which is a sequence of its own, will also be discussed in detail. Depending on the application domain and the available information, each subprocesses can be revisited if necessary. The overall process is pictured in Fig. 3. For the representation of the processes, we used the specification language BPMN,[4] which has been modified in order to fit our purpose.

The development process starts with the identification of the expectations the stakeholders have in order to derive key aspects that have to be covered by the software system. Based on the available knowledge and the insights obtained, the domain has to be separated to determine the snippet descriptions and their associated knowledge sources (from which the snippets/cases are generated). Once the snippet descriptions, snippet sources and interactions are designed, the CBR system can be implemented, evaluated, and, if necessary, incrementally improved.

4.1 Requirement Specification

This introductory phase of the system development is activated by the demand of creating a new knowledge-based system (e.g. by a customer). Furthermore, this step will be the basis for all further developments. The tasks that have to be fulfilled

[4]http://www.bpmn.org/.

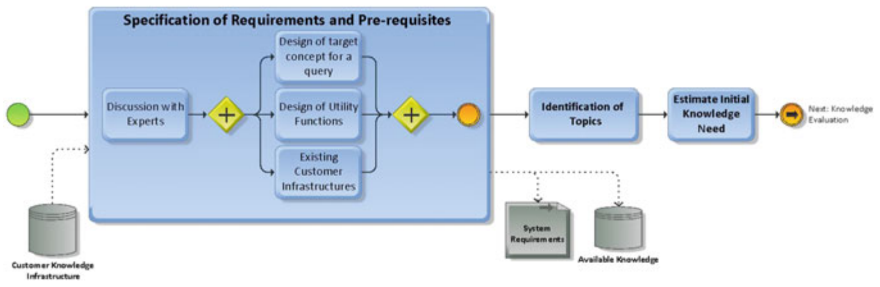

Specification of Requirements and Pre-requisites
Discussion with Experts
Design of target concept for a query
Design of Utility Functions
Existing Customer Infrastructures
Identification of Topics
Estimate Initial Knowledge Need
Next: Knowledge Evaluation
Customer Knowledge Infrastructure
System Requirements
Available Knowledge

Fig. 4 Requirement specification process

are the determination of the requirements from which the topics for the knowledge distribution can be derived in order to define the required knowledge (see Fig. 4).

The Knowledge Engineer should carry out a goal-oriented development and therefore a thoroughly requirements acquisition has to be carried out. Together with stakeholders, first goals should be identified, and based on them the requirements are iteratively refined. Relevant aspects in this phase are demanded and desired functionalities. Further, also standard factors such as numbers of expected users or the expected access.

In the end of this phase a common concepts describing the system should be available. This must contain the specification of a query and the expected prototypical result as well as information from which topic agent each sub-result should be retrieved. The latter briefly defines the required snippet descriptions, which will be recalled later.

In parallel information sources already available have to be identified and tested whether they can be included in the new system. Within an enterprise often databases or data warehouses can be accessed. If this is possible, the Knowledge Engineer has to ensure that the service will be available, required information is accessible and the provenance of information can be trusted (as well as the stakeholders trust them). For relevant information sources that will potentially feed data in a system, it has to be specified how they are structured and stored. The result of this task is a picture of the complete available knowledge.

Based on the available knowledge topics have to be identified in order to make use of the advantages of the decentralized CBR systems defined in the Knowledge Line. Each case base covers a heterogeneous topic. The definition of topics represents the main thematic areas and are directly related to the information/expectation about the system provided by stakeholders or initiators.

For each topic the required knowledge has to be estimated, in order to fulfill its task within the Knowledge Line. This estimation focuses on semantic estimations rather than the volume. This might lead to a constellation that topics have to be covered, where only little information is available. This will lead into a status of *increased knowledge demand* for this particular topic. On the other hand, if sufficient information is available, this knowledge demand can also be classified as *covered*.

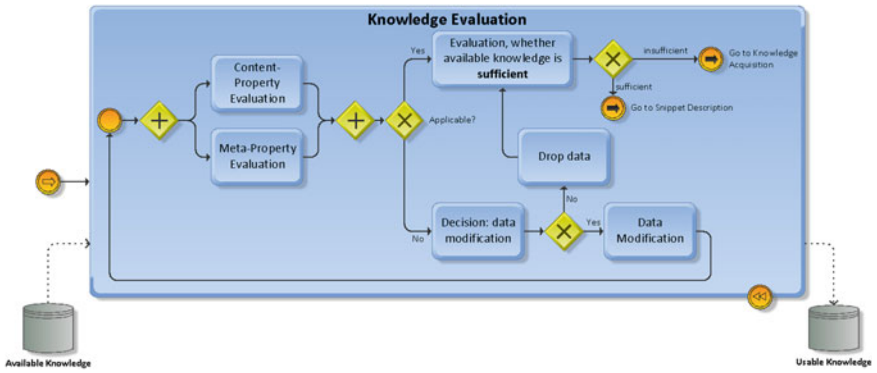

Knowledge Evaluation
Content-Property Evaluation
Meta-Property Evaluation
Applicable?
Yes
No
Evaluation, whether available knowledge is **sufficient**
insufficient
Go to Knowledge Acquisition
sufficient
Go to Snippet Description
Drop data
Decision: data modification
No
Yes
Data Modification
Available Knowledge
Usable Knowledge

Fig. 5 Evaluation of available knowledge

4.2 Knowledge Evaluation and Acquisition

Following the initial phase of collecting system requirements and elaborating available knowledge, in this phase an evaluation of that knowledge is carried out. The result of the evaluation will lead into further processes of more goal oriented knowledge acquisition for certain topics. After adding new knowledge (bases) this phase has to be carried out again in order to either identify the need of further investigation or carrying on to the next phase. The repetition of this phase is carried out until sufficient knowledge is available. Afterwards the snippet descriptions are defined.

The knowledge evaluation task (see Fig. 5) is carried out as testing and validating of the available knowledge until the previously defined goals are met. In the subsequent runs of this phase, only the newly added knowledge, which originates from the knowledge acquisition, has to be evaluated and existing knowledge does not necessarily have to be evaluated twice. More importantly, it has to be ensure by the evaluation that the newly added knowledge extends the existing knowledge and whether the required knowledge is now available.

During the evaluation it has to be ensured that the containing knowledge covers the topic adequately as well as the content is correct and up-to-date (Content-Properties). Furthermore access possibilities and restrictions as well as the used data representations have to be captured (Meta-Properties). If the evaluation results in the fact that the newly included knowledge does not fit for the topic, it has to be decided whether the new knowledge sources can/should be discarded or the knowledge has to be enhanced.

The evaluation ends with the decision whether the knowledge demand for the overall system is covered by the available knowledge sources. If there is still a lack of knowledge, the next step has to be the knowledge acquisition (see Fig. 6). If everything required is covered, it is proceeded with the identification and definition of Snippet Descriptions (see Sect. 4.3).

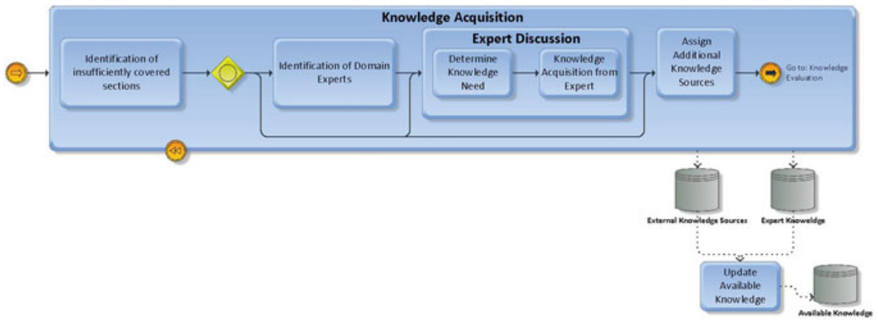

Knowledge Acquisition
Identification of insufficiently covered sections
Identification of Domain Experts
Expert Discussion
Determine Knowledge Need
Knowledge Acquisition from Expert
Assign Additional Knowledge Sources
Go to: Knowledge Evaluation
External Knowledge Sources
Expert Knoweldge
Update Available Knowledge
Available Knowledge

Fig. 6 Knowledge acquisition

While the knowledge evaluation basically only reviews the knowledge in order to detect knowledge lacks, the knowledge acquisition identifies in which topic what kind of knowledge is missing. After that identification the according expert – if available – has to be consulted. If such an expert is not available s/he has to be found or the topic can not be covered as area of expertise. In this case it is still possible to include the available information, but marked with less confidence.

If an expert has been identified, the expert will be in charge to define the lack of knowledge. During this discussion, knowledge can be directly acquired. This can be procedural or contextual knowledge that clarifies relations and enables the knowledge engineer to enrich the existing knowledge. Furthermore this discussion with the expert can lead to novel knowledge sources. Usually it is not expected that an expert will formalize and insert the required knowledge. Within SEASALT, we would expect the Knowledge Engineer to carry out this task and have an expert reviewing the knowledge as well as the final results.

The result of this subprocess is accumulated expert knowledge and knowledge sources, which will enrich the existing knowledge containers. After this step this knowledge is re-evaluated in the overall context in order to ensure that the knowledge still matches the scope of the overall system and the interaction between topic agents will be still possible.

4.3 Identification and Definition of Snippet Descriptions

Once the previous cyclic processes of knowledge evaluation and acquisition end up with the fact that the available knowledge is sufficient to represent the identified topics, relevant snippet descriptions are defined. For each snippet its representation has to be determined and based on that its correctness.

The identification of snippet description (Fig. 7) can be differentiated between descriptions derived explicitly from the system requirements and descriptions implicitly described in the available domain knowledge. Snippet descriptions which are

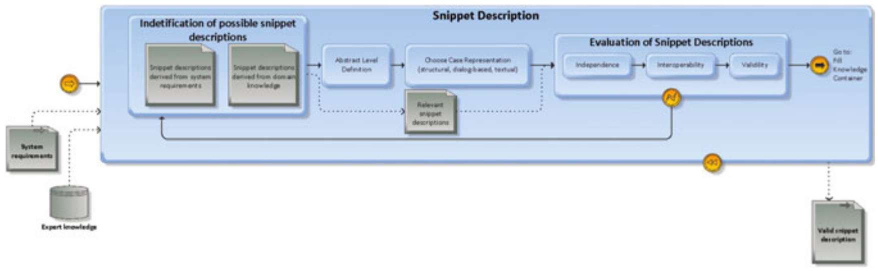

Snippet Description
Indetification of possible snippet descriptions
Snippet descriptions derived from system requirements
Snippet descriptions derived from domain knowledge
Abstract Level Definition
Choose Case Representation (structural, dialog-based, textual)
Relevant snippet descriptions
Evaluation of Snippet Descriptions
Independence
Interoperability
Validity
Go to: Fill Knowledge Container
System requirements
Expert knowledge
Valid snippet description

Fig. 7 Identification snippet descriptions

based on the system requirements usually cover desired functionalities or topics. These functionalities are incomplete and focus only on particular aspects that, from our experience, match the expert's area of work/expertise. Especially when topics are combined and the combination possibilities are discussed, more topics arise. However, each topic that comes up does not necessarily need to be represented as such in the knowledge line. The knowledge engineer has to decide whether this aspect is relevant (and therewith becomes an additional topic), it can be merged into an existing topic, or has to be withdrawn. In the end there will be a set of relevant snippet descriptions, which will the be classified as *relevant for the domain* or *additional information*. This process can therewith change the demands of knowledge for each topic, but also helps to reduce the effort, if topics are identified, which are not that relevant or can be covered differently. On the other hand, this process can also be applied if an extension of the functionalities is necessary, because it defines the demand of knowledge and the knowledge acquisition is later on carried out by the standardized process.

For each of the included snippet descriptions, the level of abstraction has to be defined. According to [12], there are three levels of abstraction for cases: *Concrete Cases* represent the real cases with very less loss of information and therewith are the lowest level of abstraction. *Abstract Cases* are reduced in their complexity. For abstract cases the loss of information is immanent and the degree of abstraction can vary as well as concrete cases and abstract cases can create a hierarchy. *Generalized Cases* represent a collection of cases with common features. Based on the level of abstraction, the kind of case representation will be defined.

After finding appropriate level of abstraction and case representation, the correctness of the therewith defined snippet descriptions has to be ensured. Each snippet descriptions has to fulfill the following domain-dependent criteria: independence, interoperability and validity.

The *independence* criterion requires that each snippet description is an individual, semantically coherent unit. This means each snippet description should cover a topic that would also work for itself and therewith each snippet also provides a logic piece of information and does not necessarily need the complete set of snippets to be semantically understandable. Further on, the *interoperability* ensures that snippets can be combined with others while the links or dependencies describe the kind of combination [3]. It is obligatory that each snippet description has to be linked at least once to another description in order to be included in the knowledge line. The interoperability herewith describes the semantic relationships between topics. The final criterion, *validity*, eventually monitors that snippets are technically combinable – it checks common representations, identifiers and constraints. To satisfy this criterion it has to be ensures that combination procedures are available and well defined.

In case one of the three above mentioned criteria is not met, either the missing information has to be added or the snippet description has to be redefined. If all requirements are satisfied, the definition of snippet descriptions is finished and the collection of valid snippet descriptions is available for the next phase - the implementation.

4.4 System Implementation

This phase first defines how the future system is populated with the acquired knowledge, before the CBR system itself is implemented (Fig. 8).

For each of the previously defined snippet descriptions one or more knowledge sources have to be assigned for the case base population. Depending on the domain, the available snippets, the Knowledge Engineer has to decide how many sources are assigned to which snippet description and how much overlap of snippets is allowed. Furthermore, in order to be able to combine information, it is necessary to include certain information in more than one snippet description - the deployment of these information is also defined and implemented within this phase.

Next, the technical procedures for the population have to be defined. This also specifies whether the knowledge to be included is stored locally in the CBR systems or queried on demand. This highly depends on the given infrastructure and availability of knowledge sources in the particular domain.

The technical procedures closely relate to the properties of a knowledge source, which have been determined in the knowledge evaluation phase. These properties define whether and automated population is possible and how data updates are fea-

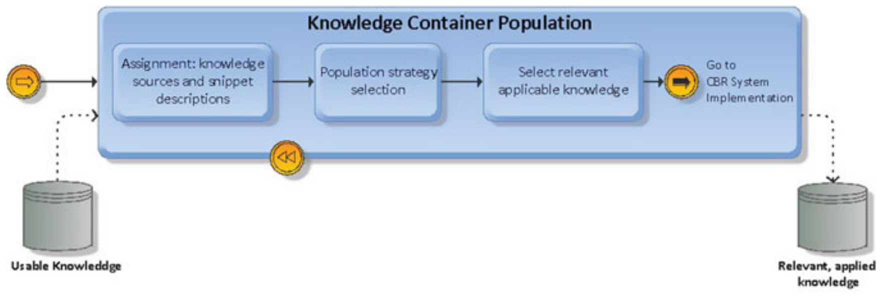

Knowledge Container Population
Assignment: knowledge sources and snippet descriptions
Population strategy selection
Select relevant applicable knowledge
Go to CBR System Implementation
Usable Knowleddge
Relevant, applied knowledge

Fig. 8 Knowledge container population

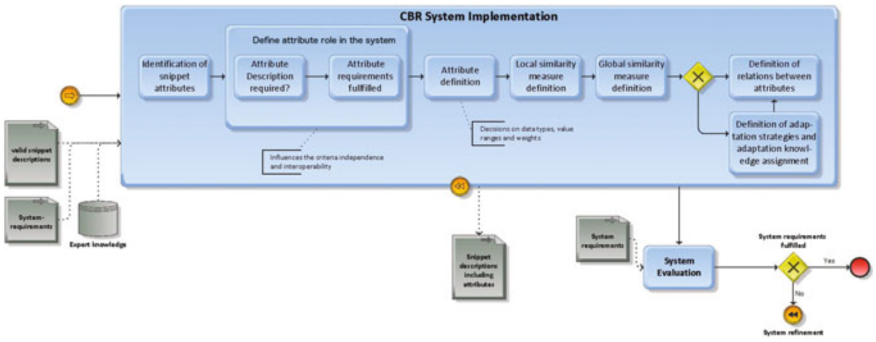

CBR System Implementation
Define attribute role in the system
Identification of snippet attributes
Attribute Description required?
Attribute requirements fulfilled
Attribute definition
Local similarity measure definition
Global similarity measure definition
Definition of relations between attributes
Definition of adaptation strategies and adaptation knowledge assignment
Influences the criteria independence and interoperability
Decisions on data types, value ranges and weights
valid snippet descriptions
System-requirements
Expert knowledge
Snippet descriptions including attributes
System requirements
System Evaluation
System requirements fulfilled
Yes
No
System refinement

Fig. 9 Implementation of the CBR system

sible. Since each snippet description is independent this has to be specified for each individual snippet description and includes access mechanisms and protocols.

After the technical specifications have been defined, the content-based selection has to follow. Not all usable information of the knowledge sources are usually relevant for a snippet. Therewith, the goal is to determine that the content of a particular snippet description fits the expectations and can be applied.

Once all relevant sources are known, accessible and the target representations are linked the implementation and population of the CBR system can start (Fig. 9). In the beginning the attribute descriptions for each snippet have to be identified in order to create a case representation. Based on the system requirements, the knowledge engineer has to decide which are the required attribute descriptions. This decision should be based on the independence criteria described in Sect. 4.3 and aim at only including necessary attributes.

After all relevant attribute descriptions are sorted out, the knowledge engineer has to analyze whether the case description is sufficient for its purpose within the application. The analysis ensures that the attribute descriptions as modular as possible and required and furthermore the attribute values can be combined during the knowledge composition. In this process, also the data types and value ranges are defined as well as the local similarity measure(s).

We assume that we have amalgamation functions to describe the global similarity. Therefore the relevance of each attribute description for the case (or snippet) has to be determined and assigned. In the final step optional adaptation mechanisms are defined. In general, the previously mentioned steps describe the process of building a CBR system once all required information is known and available.

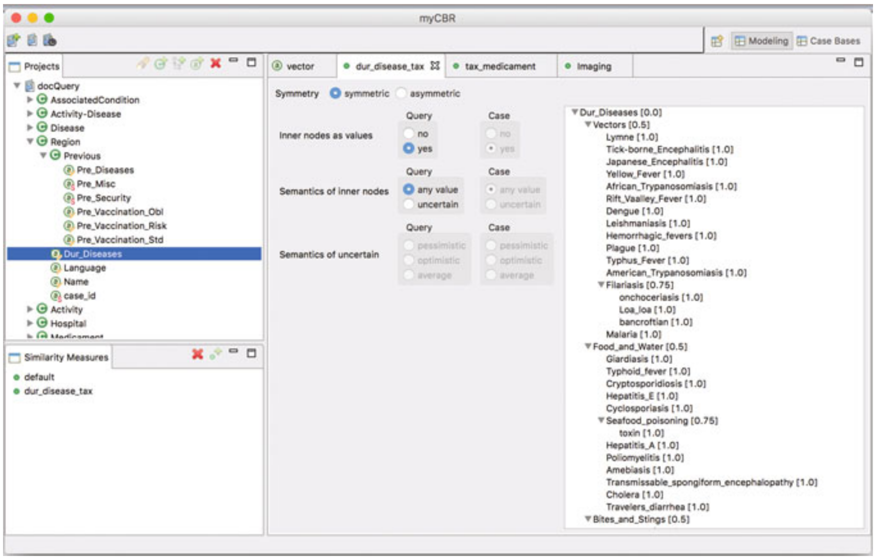
The closing step of this process is the evaluation of the created CBR system regarding the requirements. If the demands set for the system are met, the modularizing phase is completed. If there are mismatches between the expectations and the performance, the system has to be revised.

There are various rescue points where the revision can start. First the type of mismatch has to be identified, before the process can move to the according task. If there are general drawbacks regarding the incoming data, the knowledge evaluation and acquisition has to be refined (see Sect. 4.2). Alternatively, decisions made regarding the design of snippet descriptions and population processes can be revised which can influence the type of information produced by the overall system. Eventually, also the technical implementation can cause mismatches to the expectations and therewith the implementation might has to be revised as well. A detailed study of the Knowledge Line modularization methodology in a different domain can be found in [13].

5 Software Engineering of the Knowledge Line

While the Knowledge Line describes a process of how to collect and organize knowledge, it requires tools to capture and implement it in order to use it within an intelligent system. For the entire process we have successfully used myCBR [14, 15], which allows to model the knowledge as well as to test the case-based retrieval.

Figure 10 shows the myCBR view that is used to create the knowledge model for case bases. The tool supports the creation of various concepts of which each of them can hold individual case bases. Therewith it is possible to visualize the included knowledge structure along with the similarity measures during discussions with experts. Further on, the tool also allows to carry out the similarity based retrieval for each concept and therewith directly evaluate how changes made to the knowledge



```
docQuery
  AssociatedCondition
  Activity-Disease
  Disease
  Region
    Previous
      Pre_Diseases
      Pre_Misc
      Pre_Security
      Pre_Vaccination_Obl
      Pre_Vaccination_Risk
      Pre_Vaccination_Std
    Dur_Diseases
    Language
    Name
    case_id
  Activity
  Hospital
  Medicament
```

```
default
dur_disease_tax
```

Symmetry: [✓ symmetric] asymmetric

	Query	Case
Inner nodes as values	no [✓ yes]	no [✓ yes]
Semantics of inner nodes	[✓ any value] uncertain	[✓ any value] uncertain
Semantics of uncertain	pessimistic optimistic average	pessimistic optimistic average

```
Dur_Diseases [0.0]
  Vectors [0.5]
    Lymne [1.0]
    Tick-borne_Encephalitis [1.0]
    Japanese_Encephalitis [1.0]
    Yellow_Fever [1.0]
    African_Trypanosomiasis [1.0]
    Rift_Vaalley_Fever [1.0]
    Dengue [1.0]
    Leishmaniasis [1.0]
    Hemorrhagic_fevers [1.0]
    Plague [1.0]
    Typhus_Fever [1.0]
    American_Trypanosomiasis [1.0]
    Filariasis [0.75]
      onchoceriasis [1.0]
      Loa_loa [1.0]
      bancroftian [1.0]
    Malaria [1.0]
  Food_and_Water [0.5]
    Giardiasis [1.0]
    Typhoid_fever [1.0]
    Cryptosporidiosis [1.0]
    Hepatitis_E [1.0]
    Cyclosporiasis [1.0]
    Seafood_poisoning [0.75]
      toxin [1.0]
    Hepatitis_A [1.0]
    Poliomyelitis [1.0]
    Amebiasis [1.0]
    Transmissable_spongiform_encephalopathy [1.0]
    Cholera [1.0]
    Travelers_diarrhea [1.0]
  Bites_and_Stings [0.5]
```

Fig. 10 myCBR model view showing the different case bases on the *upper left*, the similarity measures for one attribute on the *lower left* and on example taxonomy in the main screen

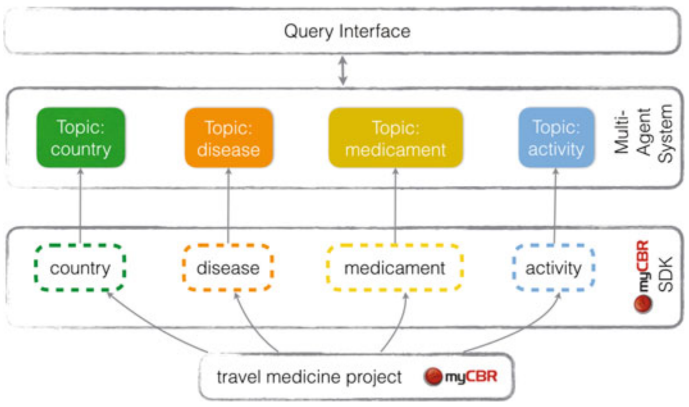


Fig. 11 myCBR-based architecture of a Knowledge Line implementation in the travel medicine domain

model and especially similarity measures affect the retrieval result. Also various case bases can be created and kept in parallel, which allows different testing scopes.

Once the CBR system is designed, the resulting project can be integrated in any kind of Java application. The myCBR back end comes as a jar file and together with the project file the entire CBR system can be run independently from the workbench.

Figure 11 shows a possible architecture of a myCBR-powered Knowledge Line application. We assume that we have one project that contains all knowledge models and case bases, each topic agent can individually be instantiated and represented by a topic agent. Numerous topic agents are coordinated by a multi-agent system (MAS) that collaboratively compose a solution. The dependencies that guide the knowledge composition process are implemented as an MAS and JADE (see [16] for details).

On top of the MAS, we suggest a query interface that receives and organizes the query process. The query interface typically has knowledge of the case representations within the MAS and can therewith provide queries for the CBR system in the right structure, while the MAS manages the internal case base dependencies. The query interface can be implemented using the Spring framework[5] which sends queries to the MAS and exposes a RESTful web API which can be used by various types of front ends.

Further on, the population of case bases for each topic can be implemented individually, so each case base creation can follow a customized process of accessing external knowledge sources, apply information extraction and natural language processing, if necessary, for feeding in knowledge snippets.

[5]https://spring.io/.

6 Summary

In this chapter we present an knowledge engineering approach that describes how a complex domain with heterogeneous knowledge components can be systematically transferred into a distributed CBR system. The resulting CBR system is a multi-agent system that utilizes several homogeneous CBR engines to hold and provide knowledge on demand. We describe the conceptual process of identifying, organizing and implementing such a knowledge-based system.

Eventually we give an overview how the concept can be implemented using existing open source tools and frameworks. To illustrate the entire process we use the travel medicine domain and discuss the challenges it provides.

References

1. Aamodt, A., Plaza, E.: Case-based reasoning: foundational issues, methodological variations, and system approaches. Artif. Intell. Commun. **7**(1), 39–59 (1994)
2. Bach, K.: Knowledge acquisition for case-based reasoning systems. Ph.D. thesis, University of Hildesheim, München (2012). ISBN 978-3-8439-1357
3. Althoff, K.D., Reichle, M., Bach, K., Hanft, A., Newo, R.: Agent based maintenance for modularised case bases in collaborative multi-expert systems. In: Proceedings of AI2007, 12th UK Workshop on Case-Based Reasoning, pp. 7–18 (2007)
4. Reichle, M., Bach, K., Althoff, K.D.: Knowledge Engineering within the Application Independent Architecture SEASALT. In: Baumeister, J., Nalepa, G.J. (eds.) International Journal of Knowledge Engineering and Data Mining, pp. 202–215. Inderscience Publishers (2011)
5. Parnas, D.L.: On the criteria to be used in decomposing systems into modules. Commun. ACM **15**, 1053–1058 (1972)
6. Tautz, C.: Customizing software engineering experience management systems to organizational needs. Ph.D. thesis, Universität Kaiserslautern (2000)
7. Bergmann, R., Althoff, K.D., Breen, S., Göker, M.H., Manago, M., Traphöner, R., Wess, S.: Selected applications of the structural case-based reasoning approach. Developing Industrial Case-Based Reasoning Applications: The INRECA-Methodology, *LNCS*, vol. 1612, pp. 35–70. Springer (2003)
8. Davenport, T.H., Prusak, L.: Working Knowledge: How Organizations Manage What they Know. Harvard Business School Press (2000)
9. Bach, K., Reuss, P., Althoff, K.D.: Case-based menu creation as an example of individualized experience management. In: Maier, R., Kohlegger, M. (eds.) Professional Knowledge Management. Conference on Professional Knowledge Management (WM-2011), From Knowledge to Action, pp. 194–203. LNI 182, Köllen Druck & Verlag GmbH, Bonn (2011)
10. Ihle, N., Newo, R., Hanft, A., Bach, K., Reichle, M.: CookIIS - A Case-Based Recipe Advisor. In: Delany, S.J. (ed.) Workshop Proceedings of the 8th International Conference on Case-Based Reasoning, pp. 269–278. Seattle, WA, USA (2009)
11. Redmond, M.: Distributed Cases for Case-Based Reasoning: Facilitating use of multiple cases. In: AAAI, pp. 304–309 (1990)
12. Bergmann, R.: Experience Management: Foundations, Development Methodology, and Internet-Based Applications. Lecture Notes in Computer Science, vol. 2432. Springer (2002)
13. Marter, S.: Case-Based Coordination Agents - Knowledge Modularization and Knowledge Composition (Fallbasierte Koordinationsagenten – Wissensmodularisierung und Wissenskomposition für dezentrale, heterogene Fallbasen). Master's thesis, Institute of Computer Science, University of Hildesheim (2011)

14. Bach, K., Althoff, K.D., Newo, R., Stahl, A.: A case-based reasoning approach for providing machine diagnosis from service reports. In: Ram, A., Wiratunga, N. (eds.) Proceedings of the 19th Intl. Conference on Case-Based Reasoning (ICCBR-2011), London, UK, LNCS, vol. 6880, pp. 363–377. Springer, Heidelberg (2011)
15. Stahl, A., Roth-Berghofer, T.R.: Rapid prototyping of cbr applications with the open source tool mycbr. In: ECCBR '08: Proceedings of the 9th European conference on Advances in Case-Based Reasoning, pp. 615–629. Springer, Berlin, Heidelberg (2008)
16. Reuss, P.: Concept and Implementation of Knowledge Line Retrieval Strategies for Modularized, Homogeneous Topic Agents within a Multi-Agent-System (Konzept und Implementierung einer Knowledge line – Retrievalstrategien für modularisierte, homogene Topicagenten innerhalb eines Multi-Agenten-Systems). Hildesheim: Stiftung Universität Hildesheim, Institut für Informatik, Bereich Intelligente Informationssysteme, Master thesis (2012)

Part II
Application Studies

Agile Knowledge Engineering for Mission Critical Software Requirements

Paolo Ciancarini, Angelo Messina, Francesco Poggi and Daniel Russo

Abstract This chapter explains how a mission critical Knowledge-Based System (KBS) has been designed and implemented within a real case study of a governamental organization. Moreover, the KBS has been developed using a novel agile software development methodology. Due to fast changing operational scenarios and volatile requirements, traditional procedural development methodologies perform poorly. Thus, an Agile-like methodology has been exploited, called iAgile. The KBS is based on an ontology used to merge different mental models of users and developers. Moreover, the ontology of the system is useful for interoperability and knowledge representation. Mission critical functionalities have been developed in 5-week cycles, along with the ontology. So, the KBS serves for three main activities: (i) requirement disambiguation, (ii) interoperability with other legacy systems, and (iii) information retrieval and display of different informative sources.

P. Ciancarini (✉)
Department of Computer Science and Engineering,
Consorzio Interuniversitario Nazionale per l'Informatica (CINI), University of Bologna,
Mura Anteo Zamboni, 7, 40126 Bologna, Italy
e-mail: paolo.ciancarini@unibo.it

A. Messina
Defense and Security Software Engineers Association,
Innopolis University, Russian Federation,
Via A. Bertoloni, 1/E – Pal.B, 00197 Rome, Italy
e-mail: segreteria@dssea.eu

F. Poggi
Department of Computer Science and Engineering, University of Bologna,
Mura Anteo Zamboni, 7, 40126 Bologna, Italy
e-mail: francesco.poggi5@unibo.it

D. Russo
Department of Computer Science and Engineering,
Institute of Cognitive Sciences and Technologies, Italian National Research Council (CNR),
Consorzio Interuniversitario Nazionale per l'Informatica (CINI),
Mura Anteo Zamboni, 7, 40126 Bologna, Italy
e-mail: daniel.russo@unibo.it

G.J. Nalepa and J. Baumeister (eds.), *Synergies Between Knowledge Engineering and Software Engineering*, Advances in Intelligent Systems and Computing 626, https://doi.org/10.1007/978-3-319-64161-4_8

1 Introduction

The most critical phase in system design is the one related to the full analysis and understanding of “User Requirements”. Difficulties arise especially where the ambiguity on the functions to implement is a continuous challenge. Due to the volatility of the user needs, changing of scenarios, and the intrinsic complexity of software products, requirement engineering benefits from an Agile approach in terms of (i) attainment with user’s contingent needs, (ii) velocity, and (iii) cost reduction.

Formal methods for requirement engineering have primarily been conceived to drive efficiently the link between customers and developers [1]. They focus on reducing the management risk connected with the initial software production phase. The results achieved by these strategies are controversial and not always cost effective [2]. The diffusion of the use of Agile practices in the software production process is putting the human factor as the key asset to capture and understand the user needs [3].

We experienced an extensive use of methodologies to identify the “unexpressed dimension” of the user requirements and to surface the “implicit” knowledge of users within a real case study of an Italian governmental Agency.

The underlying principle is the methodological formalization of the non-linear human thinking into requirements in the form of agile “User Stories”. Such an approach was successfully implemented within a mission critical organization to develop critical software applications. User stories are sentences written in natural language and have a very simple structure. The vocabulary used to write a user story depends on which user describes her need, thus is some sense it depends on the mental model the user has of her needs [4]. Capturing the essence of the users mental models [5] and overcoming the intrinsic ambiguity of the natural language are the two main goals of our study. Multiple dimensions to build a dynamic representation of requirements are the core innovative aspect of this work.

We give a problem definition of how to structure the description of user stories following Agile principles. The lessons we learned and some considerations about the importance of ontology based solutions for Knowledge Based Systems (KBS) in this context are discussed. The proposed approach is useful not only for requirement engineering but also to structure a highly interoperable knowledge representation architecture which enables a fast and flexible use in mission critical contexts.

This chapter is organized as follows. In Sect. 2 we review the most critical aspects of the use of KBS in mission critical systems; we also recall the basics of the *iAgile* process. Section 3 shows how we manage requirements using an ontology. The use of KBS technologies by the sponsoring organization is explained in Sect. 4. Finally, we draw our conclusions, summing up our findings in Sect. 5.

2 Complex Software Systems Specification

In mission critical domains, the velocity of release delivery is often considered as one of the most valuable assets. A release will usually be a partial version of the final product, but the important issue is that it already works usefully for its users. An on-field command view of a military operation (i.e., user view of a Command and Control system) typically is: "*I want the right information at the right time, disseminated and displayed in the right way, so that Commanders can do the right things at the right time in the right way*" [6].

Important functionalities may be developed or refined in the first few sprints, due to the continuous interaction between users and developers. The primary objective of this constant dialogue within the development team is the rise of the implicit and unexpressed knowledge, which will be translated by developers into software artifacts.

One typical example in mission critical contexts is the "situational awareness". It may be described as: "*The processes that concern the knowledge and understanding of the environment that are critical to those who need to make decisions within the complex mission space*" [6].

Such a sentence contains a huge quantity of implicit knowledge. For example, the interpretation of "*those who need to make decisions*" has to be clarified. More generally, in a typical agile user story words like "situational awareness" would be written as "*as the one who needs to make decisions, I want to achieve the knowledge and understanding of the environment that are critical to accomplish my mission*". This statement is, of course, still overloaded with implicit knowledge.

In our case study, this issue was overcome through a careful composition of the team including domain experts. Continuous face to face and on-line interactions allowed to minimize information asymmetry [7] and align the different mental models [5]. The main shared target was to deliver effective software to end users in a fast way.

To understand better the main use cases, consider that a military C2 Information System (IS) for mission critical purposes is essentially built on the exercise of authority and direction by a properly designated commander over assigned forces in the accomplishment of the mission [8].

In order to deliver this capability several integrations have to be taken into account, i.e., hardware, software, personnel, facilities, and procedures/routines. Moreover, such a system is supposed to coordinate and implement processes, like information collection, personal and forces management, intelligence, logistics, communication, etc. These functions need to be displayed properly, in order to effectively support command and control actions [9].

The IS we are reporting on has been based upon the development of mission specific services, called Functional Area Services (FAS), which represent sequences of end-to-end activities and events, to execute System of Systems (SoS) capabilities. These mission oriented services are set up as a framework for developing users' needs for new systems. Furthermore, mission services are characterized by geographical or climate variables, as cultural, social and operative variables, which represent functional areas or special organization issues. Mission services of the C2 system have been developed according to the NATO–ISAF CONOPS (Concept of Operations), as required by management of the governmental agency we cooperate with:

- Battle Space Management (BSM)
- Joint Intelligence, Surveillance, Reconnaissance (JISR)
- Targeting Joint Fires (TJF)
- Military Engineering - Counter Improvised Explosive Devices (ME-CIED)
- Medical Evacuation (MEDEVAC)
- Freedom of Movement (FM)
- Force protection (FP)
- Service Management (SM)

Thus a C2 system is made of a set of functional areas which in turn respond to a number of user stories.

2.1 Evolution of a Mission Critical Information System Through Agile

The mission critical information system we have studied is a Command and Control system which was capable to support on-field missions according to the NATO–ISAF's framework. The initial idea was to develop a Network Centric Warfare system (NCW) [10]. This system supports many of the operational functions defined in the contest of the NCW, according to the requirement documentation. The system has been employed in many exercises and operations and went through several tests. Today the system is serving mission critical purposes in NATO–ISAF operations e.g., the Afghanistan Mission Network.

However, several difficulties and limitations arose. The acquisitions were done according to Waterfall procedures, started in the early 2000s and went on until recently. The obsolescence of the components and related functionalities, along with the maintenance and follow-up costs connected to the Waterfall software life cycle are a big issue. Several problems are related to the impossibility to develop quickly new functionalities required by on-field personnel in a fast-changing mission critical scenario e.g., a modern asymmetric warfare. This led the use of agile software development paradigms which are supposed to overcome this crucial constraints.

Therefore, since 2014 a new "Main Command and Control System" (Main C2) to support the former system (Tactical Command and Control System or Tactical C2) has been developed. It was urgent to support the evolution of the Command and Control system, assuring a higher customer satisfaction in a volatile requirement situation. Moreover, due to budget cuts, the new system had to perform better with less resources. Costs related to both development and maintenance had to shrink rapidly.

Functional Area Services (FAS) are web-based services with a client–server architecture. Any FAS software component can be separately modified to respond to specific mission needs, as defined by users. The Main C2 has been validated in NATO exercise for the first time at CWIX 2015,[1] with positive results. Core services are build to maximize interoperability with all the relevant NATO software packages available and COTS product. Therefore, Main C2 is both flexible to implement rapidly user needs, with high interoperability of already existing systems, like the Tactical C2.

To develop it, a new methodology was implemented, applying the principles of the "Agile Manifesto" [3] to both increase the customer satisfaction and reduce software cost. After the Agency's top management decided to go Agile, there was some discussion about the method to use. There was the need to exploit Agile's values and capability but within a mission-critical environment.

Scrum was found as the most suited, since it allows a high degree of accountability [11]. This methodology is very successful in commercial environments and the most widespread Agile methodology [12]. Moreover, it was the methodology which was the best known within the Agency. Therefore, other methodologies were not really taken into consideration, even though they might have given similar results.

The teams are mixed: they include developers from the defense industry and governmental officials, based at the Agency's Headquarter in Rome. The initial production phase was extremely successful and even the start up "Sprint" (production cycle of five weeks) was able to deliver valuable software [13].

What happened was that the expectation of the Agency's stakeholder grew rapidly. From 2014 to 2016, the methodology was refined, to respond to mission and security critical needs of the operations domain. Thus, an ad hoc Scrum-like methodology was developed with the name of *iAgile*, and tested for the development of the main C2 system [14].

This methodology, depicted in Fig. 1, has been developed for critical applications, where requirements change already during the first specification and after delivering the first release. The adaptation of Scrum for the special needs of C2 software systems has also been proposed in [16].

A well known approach to analyzing ephemeral requirements consists of formalizing and prototyping the requirement specification using a suitable language, like for instance Prolog [17]. The Humphrey's Requirements Uncertainty Principle remins us that, for a new software system, the requirement (the user story) will not be completely known until after the users have used it [8]. Thus, within iAgile, Ziv's Uncertainty Principle in software engineering is applied, considering that uncertainty is inherent and inevitable in software development processes and product.

The incremental development approach enables easily any change of requirements even in the later development iterations. In our case study, due to the close interaction between the "requirement chain" i.e., from the customer to the development team, FAS were delivered with a high degree of the customer satisfaction.

The Scrum methodology developed within the Agency fully supports the change of requirements according to contingent mission needs. The traditional command

[1]www.act.nato.int/cwix.

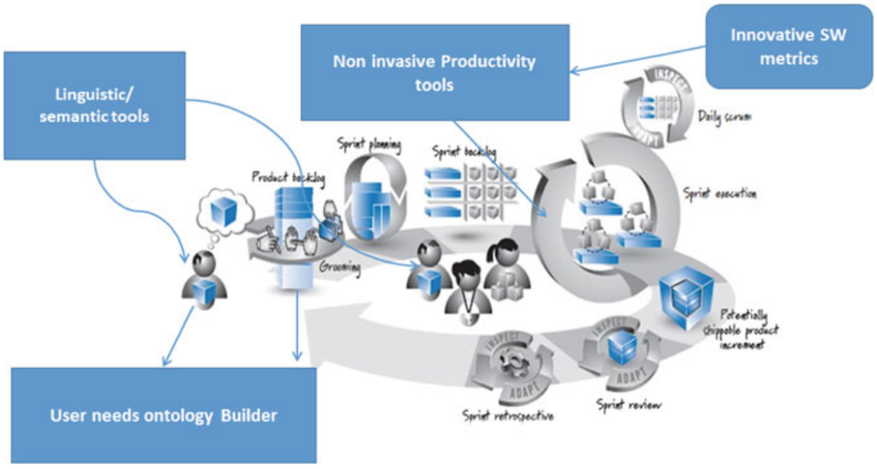


Fig. 1 Sprint representation, inspired by [15]

chain was adapted to the development needs. Both structured and horizontal characteristics of Scrum are particular effective in a critical environment. These two characteristics are embedded in the model.

Mission critical organizations need to comply with a vertical organizational chain, to empower different stakeholders to their duties. In the field where we had this experience, a hierarchy enforces clear responsibility and accountability within the command chain. So, the customer becomes the accountable official for the mission needed requirement. However, to develop different mission critical requirements, it is crucial to have a straightforward and direct communication and collaboration with final users, according to the Agile Manifesto. Therefore, in the methodology, some user representative becomes part of the development team, allowing a better understanding of the needs and a faster development of the feature.

One of the key strengths of the methodology is its flexibility. The process is defined only at a high level, to be adapted in any theater of operation. It defines values, principles, and practices focused on close collaboration, knowledge sharing, fast feedback, and tasks automation.

The main stakeholder is the Product Owner (PO), who gives to the developing team the first input which is a product's vision. It is a high level definition to address the task that will be refined during the development cycle through the Backlog Grooming.

The Backlog Grooming is a key activity which lasts over the whole development process. It focuses the problem definition, refining redundant and obsolete requirements. Moreover, it prioritizes requirements according to contingent mission needs. The acceptance criteria and the scenario definition are set by the PO in the user stories.

The developing team used by the Agency is composed as follows (such team composition is an adaptation of standard Scrum roles within a mission critical context).

- The *Product Owner* is the governmental official in charge of a specific mission critical function which has to be developed. He provides the general vision of the final functionalities to the whole team i.e., what the system has to do. It may be that PO delegates its role to another official of his team. In this case, the PO becomes a team of people that has to decided about the systems functionalities and discuss them within the development team. Ideally, the PO team has to be representative of the final user, thus it should be made also of real users.
  This crucial role is pivotal for the positive outcome of the sprint. De facto, the shortening of the "requirement chain" through the involvement of end users and the constant feedbacks of the PO during the sprint is a key success factor.
  In our case study the stakeholders were initially barely aware about the development process. Due to a constant involvement within the iterations, the stakeholders became aware of the development methodology and aligned their expectations increasing their satisfaction. Through this involvement, there is an alignment of both interests and expectations that raises the quality of the final artifact. So, the final product may not be fancy but down to earth with a high degree of immediate usability by a final user. Therefore, the degree of user involvement is of highest importance since it has a direct impact on the development itself and a ground for building a sense of ownership of the final product which is essential for the acceptation of the final product.
- The *Scrum Master* (SM) is a domain expert and is supposed to lead the development team and the Product Backlog management. The SM shapes the process according to mission's needs, leading continuous improvement like in any Agile team. He has to shield the development team from any external interferences as also to remove any hinder which may occur. What typically happens in mission critical organization is that information is shared only through very structured processes. So, there could be a loss of productivity, due the waste of time to obtain relevant information for the development process. The SM knows how to gain such information and is in charge of sharing it when needed, with no waste of time from the development side.
  According to the critical domain, he is accountable for the team's output. So, he is a facilitator but he takes the control of the team, considering also the different backgrounds of the members. Both PO and SM collaborate closely to refine requirements and get early feedbacks. Furthermore, his role is to build and sustain effective communications with customer's key staff involved in the development. Finally, he is in charge of the overall progress and take responsibility for the methodology used within the development cycles. So, he may do some corrections within the team to deliver the expected output.
- The *Development Team* composed by both military and civil contractors is in charge of the effective development. The team members are collectively responsible for the development of the whole product. Within the team there are no specialized figures (e.g., architects, developers, testers), and it is the team that organizes itself internally and takes responsibility over the entire product.
  The self organization empowers the team for the execution of the Sprint Backlog, i.e., the developed Product Backlog within the sprint, based on the prioritization

by the PO. The team members are lead by the SM who is mainly a problem solver and interfaces with the organization which needed the mission critical product. The number of team members is between three and five highly skilled software developers. The absence of a specialization is due the fact that any member is supposed to have a good knowledge about the system developed with a clear vision on the final artifact. Finally, they are also involved in the testing phase, which is carried out by an independent audit commission.

- The *Coach* is an employee of the civilian main contractor and is in charge of the management of contractual issues. Since the typical contractual form for developing contractors is body rental, the Coach facilitates organizational issues which may occur during the development cycles. Her role is to smoothen problem which may rise, to get the team oriented to the development of the artifact.

After each sprint a deployable release of the system is delivered. In order to assure security standards of mission critical applications extra testing is pursued. This activity is carried out before the deployment within the mission critical network. So, before deployment three steps are carried out as follows:

1. The development team runs a first test in the development environment and then in a specific testing environment (stage), having the same characteristics of the deployment environment.
2. Afterwards, testing activities are performed by Agency's personnel involved in test bed based activities, in limited environments to validate the actual effectiveness of the developed systems in training and real operating scenarios (Integrated Test Bed Environment).
3. Finally, testing activities on the field performed to verify the compliance of the developed systems to the national and international standards and gather operational feedback to improve the system's performance and usability.

Only after the positive check of these three steps the functionality is deployed. At the end and beginning of a new Sprint, all interested stakeholders discuss about positive and negative aspects, to improve the next iteration. Therefore, it is an incremental process, which changes with the operational scenario. It is not a frozen methodology, but it evolves along with Agency's needs.

Finally, a quite important outcome of this approach is the cost reduction in all the system's lifecycle. A first assessment of the product cost per "line of code equivalent" with respect to other comparable internally-produced software showed a cost reduction by 50%. To consider those costs we computed a comparable software by dimension (LOC) and functional area (command and control). We considered all relative cost of personnel, documentation and maintenance costs and fix cost for office's utilities. The assessment after two years showed more significant cost reduction.

Generally speaking, we know from past experiences that, on average, cost per ELOC in similar C2 domains is about 145 dollars; with regard to ground operation the cost is about 90 dollars [18]. This study, in particular, was carried out for Waterfall in a procedural context. Based on Reifer's study, we carried out our evaluation regarding iAgile cost. It was quite surprising to realize that the software we measured had an average cost of 10 dollars per ELOC.

This was possible cutting maintenance and documentation costs, which represent the most relevant part of software development costs [19]. The cost reduction came mainly from the minor rework due to requirement misunderstanding (project risk reduction) connected to the short delivery cycle and to the integration of subject matter experts into the agile teams (asymmetric pair programming typical of iAgile). Moreover, the reduction of non-developing personnel played also an important role.

Since project management responsibilities were in charge of the Agency, the use of internal personnel reduced the cost of hiring industry's senior figures. Also the increase of teams' effectiveness from sprint to sprint led to cost cuts. Due to the incremental domain knowledge acquisition gained through domain experts and user's feedbacks developers were able to produce artifacts which were attained to customer's expectation, decreasing sensibly rework.

3 Requirements Engineering, Management and Tracking

Agile software methodologies like Scrum put the development team at the center of the development process removing the major part of the procedural steps of the legacy methods and the connected "milestone evidence" mainly consisting of documents and CASE artifacts [20]. Agility is supposed to increase the production effectiveness and, at the same time, to improve the quality of the product.

However, in order to go Agile, a Waterfall-like static requirement documentation can not be replaced simply with a product backlog. The old-fashioned Waterfall frozen requirement document is no longer effective to capture the user needs in quickly changing mission critical environments. Replacing structured and consolidated text with volatile lists of simple sentences may result, in the case of complex systems, in a sensible loss of knowledge. Traceability of how the solutions are found and both the user and the developer growth may become "implicit and unexpressed knowledge" which are key elements within a high quality software development process.

Several studies suggest to overcome requirements misunderstanding as soon as possible, in order to improve the project results and to decrease development and maintenance costs within its life cycle [21]. This is one of the reason why the Agency started to develop some mission critical software in an Agile way, in order to "shorten the requirement chain", fostering software quality and cost reduction.

The ambiguity concerning the functions to implement is an everyday challenge. Due to the volatility of the user needs, changing of scenarios, and the intrinsic complexity of software products, a dynamic requirement engineering worked very well in an Agile environment [22]. However, the most challenging task is to identify the "tacit dimension" of the user requirements and to surface the "implicit" knowledge of users [23].

In most agile approaches requirements are given in the form of "User Stories", which are short sentences in natural language usually describing some value to be computed in some scenario in favor of some typical class of users. Such formalization drives non-linear human thinking in a standardized form where users have to explain how they imagine the system. This approach has been implemented for mission critical applications. Capturing users requirements and overcome the intrinsic ambiguity of the natural language are two of the main goals of this effort. Fully refined requirement specification documents are no longer meaningful; instead they should incorporate some guidelines to help the developers to effectively measure the quality of the features so that these can be improved. The result is a novel proposal based on an evolution of the "Scrum type" Product Backlog, here represented:

- *User Story*. A structured sentence which summarizes the functionality. Example:
  ```
  As <role>
  I want to <functionality description>
  in order to <goal to pursue>.
  ```
- *Business Value*. Describes the business value of the desired functionality.
- *User Story Elaboration*. It is an extended user story and it details how the functionality has to be implemented.
- *Acceptance Criteria*. Non functional requirements are given, necessary to accept the functionality (e.g., security, compliance to standards, interoperability issues). Moreover, also functional requirements have to be verified, to accept the developed software. Tests are typically focused on these functionalities.
- *Definition of Done*. It is when the story can be considered fully implemented. The Definition of Done includes the Acceptance Criteria and anything that the PO believes is necessary that the team does on the code before it can be released.
- *Expected Output*. It is a list of expected outputs from the functionality, once implemented.

Software development methodologies should be inspired by their organization's needs and not by programming concepts. Well aware of Conway's principle [24] it is the mission need that shapes the information system. Not the structure of the organization, which in our case is highly hierarchic and in its communication flows reflects the Waterfall paradigm. Due to the constant iteration between the users' community, through the Product Owner, and the development team, required applications attains users' expectations. Our experience has shown the effectiveness to overcome the limitations of existing alternatives of a Waterfall like requirement engineering, which is ineffective for complex user requirements, especially in the mission critical domain.

If continuous interaction, typical of Agile, is crucial to overcome structural information asymmetry, which is present in any human interaction [7], experience showed that it is not enough. Any software project, especially Agile, involves different people, with different backgrounds and experiences. In other terms, we all have our "mental models" [25], which are the source of this information asymmetry. Mental models are psychological representations of real, hypothetical, or imaginary situations, identified by Kenneth Craik [26]. They are mental constructs of the world around us. A mental model is a representation of the world around us and shapes our behavior and approach to problem solving. Like a pattern, once we experimented that the solution works, we tend to replicate it. It helps us to not restart from zero any time we have to face a problem. Thus, it is a simplification. So, it is a mind construct of "small-scale models" of the reality, to anticipate events, to reason, and to underlie explanation [26].

To give an example of the difference between the semantic meaning of a nominal identical concept (i.e., difference in mental models) let us consider the notion of "battle-space geometry". Starting from a user story, a PO may write: "as a commander I want to be able to represent the forward line of my sector on a map to see the deployed materials".

The user has in mind a "line" whose geometrical elements (waypoints, start, finish and type) are decided by a superior command post that is given to him as part of a formatted order packet which he expects to appear on the map by a single click of his mouse and to be updated as the tactical situation changes. The developer's first comprehension will be "drawing" tools able to produce a line by simply calling some graphic libraries. The focus is on how to implement it writing the least possible quantity of new code. This is just an example but it qualifies the differences between the two worlds very well.

For the user the image on the video is just a representation of the real world, for instance a real portion of land where the operation is taking place. Instead, for the developer the same object is the result of a series of interactions showing a map on a video where he has to provide a design tool. As trivial as they may seem these differences are at the root of the software requirement specification problem that in the past has been tackled by freezing the requirement text and operating a certain number of translations into formal representations without reconciliation of the two different mental models.

Some concepts developed in conceptual semantics explain how the representation of the world expressed in natural language is the result of a mediation between the speaker's perception of the world and the current setup of his own mind (i.e., mental models). This poses the question on what we really do communicate about requirements when we use natural language. In [27] this problem is studied, and a solution based on feature maps is proposed.

In our case what emerged is a common ontology used by both users and developers. We found out that working on the ontology in the initial production process (i.e., Product Backlog) improved the effectiveness of the Agile approach. In fact, the development of a Command and Control ontology, useful as knowledge representation tool as described in the next section, is also effective to merge different mental models and to support requirements traceability [28].

4 Use of KBS and OBS Within iAgile

The use of Ontology-Based Systems (OBS) for managing requirements and user stories when applying Agile methods has been explored many times, but it is still an unresolved issue [29, 30]. Some literature suggests that ontology driven development should be the norm, both in general and specifically in the Agile arena [31].

The use of OBS is of paramount importance in a mission critical context. We have experienced it in peace-keeping operations, where rapid information flows coming from different actors (e.g. military, NGOs, citizens, press.) have to be processed. The different needs, contexts, and objectives of these actors are often reflected into a wide range of viewpoints and assumptions, producing different, overlapping and/or mismatched concepts and structures that essentially concern the same subject matter.

Different "organizational routines" [23] lead to different communication standards, along with "tacit knowledge" [32]. Thus, there is the need to organize the different "mental models" [5] around the development process. Ontologies are a powerful tool to overcome this lack of a shared understanding, providing an unifying framework for the different viewpoints and mental models that coexist in vast and heterogeneous contexts.

As described in [28], this shared understanding provides many practical benefits, such as serving as the basis for human communication (reducing conceptual and terminological confusion), improving the interoperability among systems with different modeling methods, paradigms, languages and software tools, and supporting the main system engineering principles (such as reusability, reliability, requirements identification, system specification, etc.). The adoption of ontologies as a core components of software architectures [33] in conjunction with Agile methodology development principles has proven its effectiveness in changing and variable contexts.

An overall idea of the main elements of the development process is depicted in Fig. 2. Both KBS and OBS are build on users' mental models. This means that requirements and the ontology represent user's view and needs. So, the user stories collected are the core elements. Following Agile methodologies principles, such artifacts are fundamentals to distill both information about the system to develop, and knowledge on the domain in which such system is expected to operate. User stories have been used to extract the requirements of the C2 system, and to develop an ontology for representing the main concepts of the mission critical domain.

So, the backlog grooming (i.e., the refinement of the user stories) becomes the instant where users stories split. Requirements are defined and the ontology is developed. This split is not straightforward. Considering that ontology's entities definition is very helpful to define better user's expectations, requirements documents are developed separately from the ontology. While requirements are developed manually by the development team, the ontology is developed by Protégé[2] [34]. As shown in Fig. 3, developers already exposed to semantic technologies use this standard tool to develop the domain specific ontology.

[2]http://protege.stanford.edu.

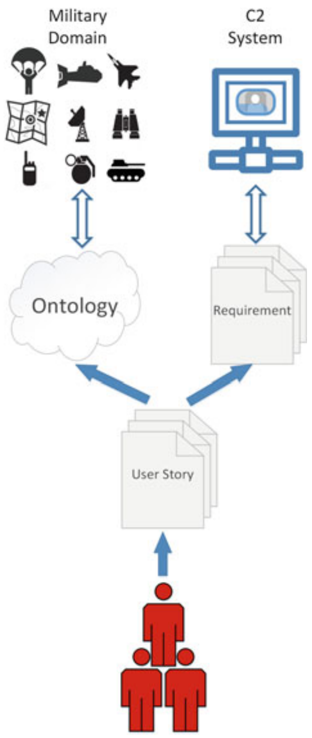


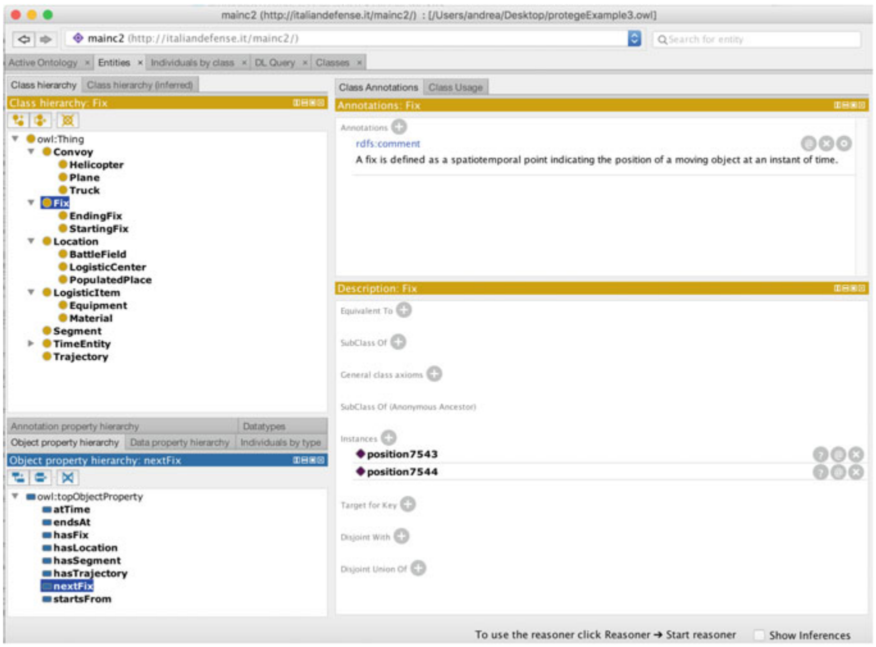
Fig. 2 The ontology of the application domain and the system requirements are derived from user stories

4.1 An Ontology-Based Architecture for C2 Systems

One of the main challenges in the mission critical domain is the ability of managing in a precise and accurate way the complexity, variability and heterogeneity of information. In particular, the ability of integrating different sources of information, extracting the most relevant elements and putting them into the context is of paramount importance for supporting the tasks of control and decision making. In our approach, ontologies and related technologies are the main tools for facing both the methodological and technological aspects of such context.

Similarly to other scenarios, also in the mission critical domain people, organizations, and information systems must communicate effectively. However, the various needs, contexts and objectives are often reflected into a wide range of viewpoints and assumptions, producing different, overlapping and/or mismatched concepts and structures that essentially concerns the same subject matter. Ontologies are often used to overcome this lack of a shared understanding, providing an unifying framework for the different viewpoints that coexist in vast and complex C2 systems.

As described in [28], this shared understanding provides many practical benefits, such as serving as the basis for human communication (reducing conceptual and terminological confusion), improving the interoperability among systems with



A fix is defined as a spatiotemporal point indicating the position of a moving object at an instant of time.

Fig. 3 A snapshot of the C2 ontology during its development with Protege. The class and property hierarchies are shown on the *left*, while other contextual information (e.g. annotations, instances and relevant properties) are shown on the *right*

different modeling methods, paradigms, languages and software tools, and supporting the main system engineering principles (such as reusability, reliability, requirements identification, system specification).

The central role played by ontologies is summarized by Fig. 4, which depicts the overall architecture of the C2 system. The ontology we have developed describes the data model contained in a Knowledge Base (KB), which contains all the information needed by the C2 system. The KB is populated by ad hoc software components (i.e. adapters), that extract information from all the different sources (e.g. legacy tactile systems, news or ONG CMS, etc.), and convert it in semantic statements that conforms to the ontology. The frequency of the KB populating processes varies from sources to sources. Where possible, triggering mechanisms have been used to identify modifications in the sources and activating the data extraction. Otherwise, adapters performs the data extraction at fixed (and configurable) intervals. Of course, high data quality is a major issue here [35].

The KB contains provenance information defining:

- the origin of the data;
- the agent (usually a software component) responsible for the data extraction;
- other useful metadata (e.g. time information, type - such as insertion, deletion, update - of the KB modification, etc.).

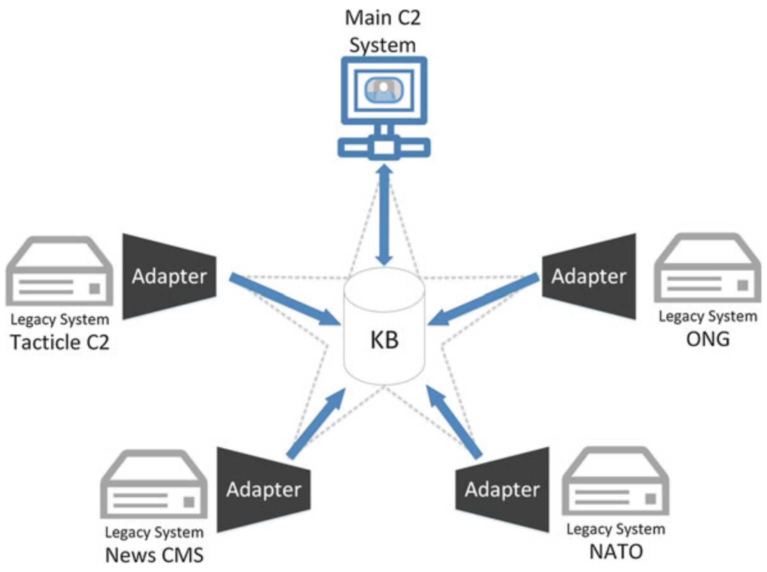


Fig. 4 The C2 system is modeled around the star architecture pattern. The domain ontology is the center of such architecture, and is used to integrate different resources and systems

The main advantages of having such information are the ability to discern among different authority levels, the capability of performing comparisons and advanced filters, just to cite a few. Moreover, from a technical point of view, provenance information are of paramount importance to have full control over the KB state during the whole system lifecycle (from the inception and development phases to the final operating period).

The Main C2 system is depicted at the upper vertex of the star architecture. It implements all the functionalities required by the users and described in the collected user stories. All the data required to expose such functionalities to the users are retrieved from the KB by means of standard semantic queries. The same mechanism is used for adding new information in the KB (e.g. for keeping track of the output of the system users' analysis and decision processes).

4.2 Developing Domain Ontologies from User Stories with iAgile

Ontologies play a crucial role in the development of the framework. The primary objective is to develop an ontology that is capable of modeling the complexity of mission critical domains. The starting point and primary source of information for this task is the set of 600 user stories collected from the system final users and domain

US 2825 (2656)
As <system user>I want <to figure out if the track relative to an object (equipment) represented on the map has been updated, or if the position has just been changed.><Such a status would be highlighted by a green border around the icon on the object. This function should be enabled or disabled by the user.>

US 2828(2659)
As <commander of the logistics>I want <to view a summary, in both tabular and graphical format (eg. histogram), of the efficiency logistic items. ><The total of logistics items and the level of efficiency should be displayed in percentage of the total.>

US 2829(2660)
As <commander of the logistics>I want <to be able to view on a geographical map the geographical distribution of identified logistical items (e.g. equipment, materials, etc.).><The map shall display the concentrations of materials at the logistic centers, represented by a specific colored icon associated with the type of material. The color should reflect the overall efficiency of the logistic item. For each logistic center, it should be possible to view the amount items, with an histogram graph showing both total items along with the percentage of efficiency.

US 2822(2653)
As <system user>I want <that the system can receive and display information about convoy (e.g. trucks, helicopters, planes, etc.) - for example Moving Convoy, Halted Convoy, etc.>

US 2821(2652)
As <system user>I want <that the system can store information about convoy (e.g. Moving Convoy, Halted Convoy, etc ...).>

US 2820(2651)
As <FAS web user>I want <to be able to view on a geographical map the geographical distribution of identified logistical items.><The map shall display the concentrations of materials at the logistic centers, represented by a specific colored icon associated with the type of material. The color should reflect the overall efficiency of the logistic item. For each logistic center, it should be possible to view the amount items, with an histogram graph showing both total items along with the percentage of efficiency.>

US 2819(2645)
As <FAS web user>I want <to be able to view on a geographical map the geographical distribution of identified logistical items.><The map shall display the concentrations of materials at the logistic centers, represented by a specific colored icon associated with the type of material. The color should reflect the overall efficiency of the logistic item. For each logistic center, it should be possible to view the amount items, with an histogram graph showing both total items along with the percentage of efficiency.>

US 2711(2296)
As <system user>I want <to display on the map the trajectory and the last positions of a specific object. This functionality should be enabled or disabled by the user.>

Fig. 5 User stories collected with iAgile are used to develop the system domain ontology

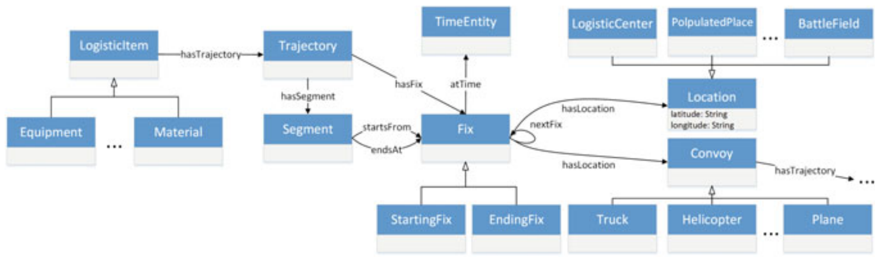


Fig. 6 A fragment of the developed domain ontology. Three general concepts are represented (i.e. logistic item, location and convoy), and the trajectory pattern has been used to model positional information

experts. User stories have been grouped in small buckets (having between 5 and 10 user stories each). The process started by considering a first bucket, that has been used to develop an initial model. Finally, the ontology has been developed iteratively, by adding a new user story bucket to the set of already considered ones at each cycle.

At the end of each cycle, a small dataset (*test dataset*) is created according to the current ontology and considering the user stories under examination. Such a dataset is used to perform a *quality check* on the current ontology. In practice, tests are a series of queries that are derived by analyzing the functionalities described in the user stories. A query test must be executed on the test dataset to check the ontology validity after the modifications performed in the last cycle. In case of a positive result, a new user story bucket is considered and another development cycle is performed. Otherwise, the ontology is refactored and modified until the quality check is satisfied.

In order to clarify the ontology development process, we present in Fig. 5 a bucket composed by eight real user stories collected for the C2 system.

Each user story is identified by a unique identifier. User stories have a common fixed structure, where users' roles, objectives and user story specifications are easily identifiable. During a further step of analysis, the concepts that are the main candidates for becoming classes in the domain ontology have been underlined.

An excerpt of the ontology developed starting from the eight user stories is depicted in Fig. 6. Three general concepts are modeled: logistic item, location and convoy. Well known ontology engineering principles and best practices described in [36] have been used throughout the whole process. As shown in the ontology fragment, for example, the trajectory pattern [37] has been used to model objects positions and movements. This is an elegant solution for attaching trajectories composed of segment with a geographical or physical extents to any object of the domain.

The process of ontologically modeling the domain has many practical benefits. For example, we successfully converted requirements and constraints to the data model in semantic assertions. Such restrictions can be automatically checked by using popular semantic tools (e.g. reasoners). For instance, we can imagine to add an assertions that states the segments of a trajectory should not be in overlap with respect of both the time and space dimensions. In another words, we can impose that an object should not have two different positions at the same time. All the cases that do not respect such limitations in the data are automatically identified as inconsistencies stated by the semantic reasoners.

5 Conclusions

Knowledge Based Systems and Ontology Based Systems are key elements for the development of software-intensive mission critical systems. We reported about a real case study concerning a mission critical system developed for an Italian governmental Agency. Volatile requirements and fast changing operational scenarios led to the choice of a new development process model, transitioning from Waterfall to Agile. However, Agile is not a *panacea per se* but needs to be adapted for complex mission critical purposes. We customized Scrum into *iAgile*, developing a flexible but structured paradigm. Ad hoc steps were designed to comply with both velocity and security. Moreover, we found that such methodology led to an important saving of development and maintenance costs.

The role of the KBS we have used is double: to disambiguate requirements and to build an ontology for interoperability and knowledge representation. The ontology was designed using semantic tools during the requirement specification. Moreover, at the same time, the user story elaboration is carried out for the functionality development. This process allowed to align the different mental models of users and developers. Therefore, after the first formalization of entities, it supported the next Sprint backlogs with a high relevance to users' expectations.

The big advantage of the use of an ontology derives from the interoperability with other legacy systems. In a real-world operational scenario, the mission critical information system is fed by information of different provenance. As shown, also non-governmental actors may deliver useful elements for an up to date situational awareness. So, from different sources it is possible gather data in a flexible and incremental way improving the information completeness, necessary to take mission critical decisions. Moreover, through the relationship between ontologies of other governmental or intergovernmental agencies both interoperability or replacement of such system is highly simplified. Since the Agency is supposed to offer security services in multilateral and multinational operations, interoperability is of strategic importance. Therefore, the presented approach is an important driver for a smooth and effective system deployment in a mission critical environment.

Future research will go in several directions.

A comprehensive approach to OBS, based on the acquired experience, will be implemented within the Agency with the aid of knowledge-based tools. Moreover, a Machine Learning approach in which requirements are automatically processed to assist the continuous development with Scrum [38] has also to be developed. Also, the use of concurrent development methodologies to support velocity and reliability has to be improved [39], along with a flexible system's architecture. Especially, the reliability in terms of systems "antifragility" of mission critical applications needs further investigation [40, 41]. Although we are aware of the relationship between the software quality dimension and its architecture [42], still efforts need to be pursue to figure out how Agency's business goals (e.g., velocity, cost reduction) impact on the system. Finally, also issues related to software reuse have to be explored [43, 44], since the cloning practices, also in critical systems, is quite common.

Acknowledgements The Authors wish to thank the Consorzio Interuniversitario Nazionale per l'Informatica (CINI) for the partial support in the context of the AMINSEP project. We also thank CNR for supporting the authors Ciancarini and Russo.

References

1. Easterbrook, S., Lutz, R., Covington, R., Kelly, J., Ampo, Y., Hamilton, D.: Experiences using lightweight formal methods for requirements modeling. IEEE Trans. Softw. Eng. **24**(1), 4–14 (1998)
2. Lucassen, G., Dalpiaz, F., van der Werf, J., Brinkkemper, S.: Improving agile requirements: the quality user story framework and tool. Requir. Eng. **21**(3), 383–403 (2016)
3. Alliance, A.: Agile manifesto **6**(1) (2001). http://www.agilemanifesto.org
4. Thamrongchote, C., Vatanawood, W.: Business process ontology for defining user story. In: IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS), pp. 1–4. Japan (2016)
5. Porac, J.F., Thomas, H.: Taxonomic mental models in competitor definition. Acad. Manag. Rev. **15**(2), 224–240 (1990)
6. Bearden, J.B.: Command and control enabling the expeditionary aerospace force. Technical report, DTIC Document (2000)
7. Akerlof, G.: The market for lemons: quality uncertainty and the market mechanism. Essential Readings in Economics, pp. 175–188. Springer, Berlin (1995)
8. Sutherland, J.: Agile can scale: inventing and reinventing scrum in five companies. Cut. IT J. **14**(12), 5–11 (2001)
9. Staff, C.: Interoperability and supportability of information technology and national security systems. Technical report, CJCSI 6212.01E, Department of Defence (United States of America) (2008)
10. Alberts, D.S., Garstka, J.J., Stein, F.P.: Network centric warfare: developing and leveraging information superiority. Technical report, DTIC Document (2000)
11. Schwaber, K.: Agile Project Management with Scrum. Microsoft Press, Redmond (2004)
12. VersionOne: 11th annual state of agile survey (2016). http://stateofagile.versionone.com/
13. Cotugno, F.R., Messina, A.: Adapting scrum to the Italian army: methods and (open) tools. In: IFIP International Conference on Open Source Systems, pp. 61–69. Springer, Berlin (2014)
14. Messina, A., Fiore, F., Ruggiero, M., Ciancarini, P., Russo, D.: A new agile paradigm for mission-critical software development. J. Def. Softw. Eng. (CrossTalk) **6**, 25–30 (2016)
15. Rubin, K.S.: Essential scrum: a practical guide to the most popular agile process. Addison-Wesley, Upper Saddle River (2012)
16. Harvie, D., Agah, A.: Targeted scrum: applying mission command to agile software development. IEEE Trans. Softw. Eng. **42**(5), 476–489 (2016)
17. Sterling, L., Ciancarini, P., Turnidge, T.: On the animation of not executable specifications by prolog. Int. J. Softw. Eng. Knowl. Eng. **6**(1), 63–87 (1996)
18. Reifer, D.: Industry software cost, quality and productivity benchmarks. DoD SoftwareTech News **7**(2), 3–19 (2004)
19. Pressman, R.S.: Software Engineering: A Practitioner's Approach. Palgrave Macmillan, New York (2005)

20. Benedicenti, L., Cotugno, F., Ciancarini, P., Messina, A., Pedrycz, W., Sillitti, A., Succi, G.: Applying scrum to the army: a case study. In: Proceedings of the 38th International Conference on Software Engineering Companion, pp. 725–727. ACM (2016)
21. Boehm, B., Basili, V.R.: Software defect reduction top 10 list. Computer **34**(1), 135–137 (2001)
22. Gazzerro, S., Marsura, R., Messina, A., Rizzo, S.: Capturing user needs for agile software development. In: Proceedings of 4th International Conference in Software Engineering for Defence Applications, pp. 307–319. Springer, Berlin (2016)
23. Nelson, R.R., Winter, S.G.: An Evolutionary Theory of Economic Change. Harvard University Press, Harvard (2009)
24. Conway, M.: How do committees invent. Datamation **14**(4), 28–31 (1968)
25. Johnson-Laird, P.N.: Mental Models: Towards a Cognitive Science of Language, Inference, and Consciousness, vol. 6. Harvard University Press, Harvard (1983)
26. Craik, K.: The nature of exploration (1943)
27. Itzik, N., Reinhartz-Berger, I., Wand, Y.: Variability analysis of requirements: considering behavioral differences and reflecting stakeholders. IEEE Trans. Softw. Eng. **42**(7), 687–706 (2016)
28. Uschold, M., Gruninger, M.: Ontologies: principles, methods and applications. Knowl. Eng. Rev. **11**(02), 93–136 (1996)
29. Kumar, M., Ajmeri, N., Ghaisas, S.: Towards knowledge assisted agile requirements evolution. In: Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering, RSSE'10, pp. 16–20. ACM, New York (2010)
30. Machado, J., Isotani, S., Barbosa, A., Bandeira, J., Alcantara, W., Bittencourt, I., Barbosa, E.: Ontosoft process: towards an agile process for ontology-based software. In: 49th Hawaii International Conference on System Sciences (HICSS), pp. 5813–5822. IEEE (2016)
31. Knublauch, H.: Ramblings on agile methodologies and ontology-driven software development. In: Workshop on Semantic Web Enabled Software Engineering (SWESE), Galway, Ireland (2005)
32. Polanyi, M.: The tacit dimension (1966)
33. Ciancarini, P., Presutti, V.: Towards ontology driven software design. Radical Innovations of Software and Systems Engineering in the Future, pp. 122–136. Springer, Berlin (2004)
34. Knublauch, H., Fergerson, R.W., Noy, N.F., Musen, M.A.: The protégé OWL plugin: an open development environment for semantic web applications. In: International Semantic Web Conference, pp. 229–243. Springer, Berlin (2004)
35. Ciancarini, P., Poggi, F., Russo, D.: Big data quality: a roadmap for open data. In: 2nd IEEE International Conference on Big Data Service (BigDataService), pp. 210–215. IEEE (2016)
36. Gomez-Perez, A., Fernández-López, M., Corcho, O.: Ontological Engineering: With Examples from the Areas of Knowledge Management, e-commerce and the Semantic Web. Springer Science & Business Media, New York (2006)
37. Hu, Y., Janowicz, K., Carral, D., Scheider, S., Kuhn, W., Berg-Cross, G., Hitzler, P., Dean, M., Kolas, D.: A geo-ontology design pattern for semantic trajectories. In: International Conference on Spatial Information Theory, pp. 438–456. Springer, Berlin (2013)
38. Russo, D., Lomonaco, V., Ciancarini, P.: A machine learning approach for continuous development. In: Proceedings of 5th International Conference in Software Engineering for Defence Applications. Springer, Advances in Intelligent Systems and Computing (2017)
39. Russo, D.: Benefits of open source software in defense environments. In: Proceedings of 4th International Conference in Software Engineering for Defence Applications. Advances in Intelligent Systems and Computing, vol. 422, pp. 123–131. Springer, Berlin (2016)
40. Russo, D., Ciancarini, P.: A proposal for an antifragile software manifesto. Procedia computer science. In: The 7th International Conference on Ambient Systems, Networks and Technologies (ANT 2016), vol. 83, pp. 982–987 (2016)
41. Russo, D., Ciancarini, P.: Towards Antifragile Architectures. Procedia Computer Science. In: The 8th International Conference on Ambient Systems, Networks and Technologies (ANT 2017), vol. 109, pp. 929–934 (2017)

42. Russo, D., Ciancarini, P., Falasconi, T., Tomasi, M.: A software quality concerns in the Italian bank sector: the emergence of a meta-quality dimension. In: Proceedings of the 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP), pp. 63–72, IEEE (2017)
43. Ciancarini, P., Russo, D., Sillitti, A., Succi, G.: Reverse engineering: a European IPR perspective. In: Proceedings of the 31st Annual ACM Symposium on Applied Computing, pp. 1498–1503. ACM (2016)
44. Ciancarini, P., Russo, D., Sillitti, A., Succi, G.: A guided tour of the legal implications of software cloning. In: Proceedings of the 38th International Conference on Software Engineering Companion, (ICSE-SEIS), pp. 563–572. ACM (2016)

Knowledge Engineering for Decision Support on Diagnosis and Maintenance in the Aircraft Domain

Pascal Reuss, Rotem Stram, Klaus-Dieter Althoff, Wolfram Henkel and Frieder Henning

1 Introduction

The diagnosis of machines belonging to technical domains demands careful attention: dozens of relations between individual parts have to be considered and operational or environmental conditions can effect measurable symptoms and diagnoses. An aircraft is one of the most complex machines built by humans and therefore the diagnosis and maintenance of aircrafts requires intelligent and efficient solutions.

Finding the root cause for an occurred fault is sometimes difficult to reproduce or isolate, because a single part, the interaction between parts, the communication infrastructure, or even environment characteristics like temperature or pressure may cause a fault. For example, if a monitor does not display the status of a system, the monitor could be broken, one or more parts of the system sending ambiguous or no information, or the communication cable to the monitor may be broken. In addition, low temperature may cause a freeze to a cable or relay, which then may cause the defect. Therefore, the use of experience from past faults can be very helpful to get a quick and precise diagnosis and reducing the time for finding and repairing the

P. Reuss (✉) · K.-D. Althoff
Intelligent Information Systems Lab, University of Hildesheim, Hildesheim, Germany
e-mail: reusspa@uni-hildesheim.de

K.-D. Althoff
e-mail: klaus-dieter.althoff@dfki.uni-kl.de

P. Reuss · R. Stram · K.-D. Althoff
Competence Center Case Based Reasoning, German Center for Artificial Intelligence, Kaiserslautern, Germany

W. Henkel
Airbus Operations GmbH, Kreetslag 10, 21129 Hamburg, Germany

F. Henning
Lufthansa Industry Solutions, Hamburg, Germany

G.J. Nalepa and J. Baumeister (eds.), *Synergies Between Knowledge Engineering and Software Engineering*, Advances in Intelligent Systems and Computing 626, https://doi.org/10.1007/978-3-319-64161-4_9

root cause. While designing and developing such a diagnosis system, software engineering and knowledge engineering are closely related to each other. Requirements engineering and quality testing are used to design and test the software agents and processes, while knowledge engineering is required to model the CBR systems.

In this chapter, we describe a decision support system for the diagnosis and maintenance in the aircraft domain that integrates experience into the existing diagnosis approach. In the next section we give an overview of the aircraft domain and the existing diagnosis approach. Section 2 describes related research and Sect. 3 describes the OMAHA research project, the multi agent system and our framework for free text processing in detail. Section 4 gives an overview of the current implementation and test runs as well as the planned evaluation. The last section gives a summary and an outlook to future work.

1.1 Aircraft Domain

The aircraft domain is a very complex technical domain. An aircraft consists of hundreds of components (e.g., Communication and Ventilation Control), which consist of dozens of systems (e.g., Cabin Intercommunication System and Air Conditioning), that in turn contain dozens of individual parts (e.g., Flight Attendant Panel and Cabin Air Filter) called Line Replaceable Units (LRU). Not all of these components need to be monitored. For an A380 around a hundred systems are monitored with 60.000 possible faults. These systems and LRUs are interacting with and rely on each other. Therefore, it is challenging to identify the root cause of an occurred fault, because it can either be found within a single LRU, within the interaction of several components of a system, within the interaction of LRUs of different systems, or even within the communication infrastructure of different LRUs. Finding cross-system root causes is a very difficult and resource expensive task.

The existing diagnosis system onboard an aircraft can track root causes based on causal rules defined for the LRUs. These rules are not always unambiguous, because the diagnosis approach is effect-driven. Based on a comprehensible effect (visible, audible, or smellable) in the cockpit, the cabin, or the cargo bay, the diagnosis system tries to determine the system behavior that belongs to the effect and traces the root cause back through the defined rules. Based on the error messages and the identified root causes, so-called PFR items are created. A Post Flight Report (PFR) contains an average of six error messages (PFR items) for an A380, that may have several possible root causes. The PFR items are subdivided into four classes: cockpit effect, cabin effect, maintenance effect, and faults without effect. Faults within the fourth class have not to be repaired immediately, but may have an impact on future faults or scheduled maintenance. For each error messages up to three LRUs could be accused. The LRUs are ranked by their fault probability. The line mechanic replaces the accused LRUs until the fault disappears. But the LRU ranked on the first position is not always responsible for the fault. LRUs that are replaced, but working correct, cause unnecessary costs. In the worst case none of the accused LRUs are responsible for the fault. In this case the repair task will get difficult and consumes resource and time. The challenge is to find and repair the fault during the turn around time between the landing and passenger release and the boarding of new passengers. The more faults in this unscheduled maintenance have to be repaired, the greater is the risk to cause a delay of the aircraft or even an aircraft swap.

With this in mind, we are developing a decision support system based on multiple software agents and CBR systems. Within the OMAHA project the MAS will be used within a demonstrator that simulates faults in an aircraft and generates an PFR. For each item in the PFR a request to our MAS will be sent. Another use case is the customer service for Airbus. The MAS could be used to find similar cases to customer requests, reduce the response time for a solution and increase the precision of a diagnosis.

2 Related Work

Decision support for diagnosis and maintenance is an interesting and diverse field. Several paradigms were used in the past, including CBR, to create assisting systems in many fields. In Jackson et al. a fault diagnosis and prognosis system, DAME, is described, where grid computing was used [1]. A hybrid approach of model-based reasoning and CBR has also been described by Feret et al. [2].

Our approach combines multi-agent systems (MAS) with CBR, in order to improve the cost-benefit ratio. The MAS allows to distribute actions among the agents, including several CBR agents, in order to reduce computation time and effort. Corchado et al. describe their approach for using MAS in their architecture for an ambient intelligent environment named FUSION@ [3]. This work differs from our approach in domain and structure. While the SEASALT architecture defines distinct agents and their tasks, FUSION@ is more liberal in what tasks the agents can perform. Zouhair et al. describe a dynamic CBR system in the learning assistance domain that uses MAS and learns from traces left by the users [4]. Our system, however, uses technical knowledge instead of traces.

The data sources that are used in the OMAHA project are mainly structured, and this is why we chose the structural CBR approach. However, many times the most important information may be presented in the fault description and log book entries, which constitutes of one or two sentences of free text. Many systems that deal with textual data sources adopt the textual CBR approach, including [5–7]. We decided to use a hybrid approach of structural CBR and NLP techniques in order to best represent the technical knowledge at our disposal.

There are several tools and frameworks for NLP, such as Stanford CoreNLP [8], Apache Lucene [9], GATE [10], DARE [11], and most recently SyntaxNet [12]. These tools cover several algorithms, tasks, and goals in NLP, and some of them are even used in our system. Our framework combines them along with techniques from association rule mining and CBR, while extending them with in-house developed techniques that can be used directly in CBR systems.

A similar case to ours is Sizov et al. who created a CBR system to analyze aircraft accidents from investigation reports [13]. Here their main objective was to give an explanation of the causes of the accident, rather than produce a diagnosis. They transformed their textual data into text reasoning graphs for further analysis. Our system's main objective is to diagnose a fault and assist in finding a solution. Also, due to our choice in structural CBR, we attempt to extract attribute-value type of information from our data.

Another very similar work is Bach et al., who described a CBR fault diagnosis system in the automobile domain [14]. Here the same SEASALT architecture was used, along with a structural CBR approach. However, when faced with the problem of extracting information from text, even though using similar extraction methods to ours, all the terms were evaluated by experts, and the local similarity measure for symbolic attributes was manually organized into taxonomies and similarity tables. The importance for each attribute in the global similarity measure was manually assigned by experts as well. In our system, although we rely heavily on experts, we supplement the information they provide us with an automated concept extraction and similarity assignment. The importance of the attributes in the global similarity measure is completely automated in our system by using sensitivity analysis.

The progression and development of our approach has been well documented. The MAS, as thoroughly described in [15, 16], uses the SEASALT architecture, and defines specific roles for each agent type. This allows an integration of several services, most notably the CBR systems. The modeling and extraction of knowledge in the OMAHA project is done with the help of a workflow, described in [17] and later developed into the framework FEATURE-TAK [18]. It supports the automatic transformation of structured and semi-structured feature information into knowledge. The framework part regarding the attribute importance learning method has also been detail in [19].

3 Knowledge Engineering for Decision Support in Diagnosis and Maintenance

This section describes the multi-agent system for decision support in diagnosis and maintenance realized within the OMAHA project. We will describe the use cases, the concepts of the diagnosis and knowledge engineering workflow, as well as the knowledge modeling of the CBR systems and the implementation of the software agents.

3.1 Use Case and Problem Description

The current diagnosis approach for Airbus aircraft is effect-driven. A Central Maintenance System (CMS) correlates failure messages from LRUs to effects like red blinking lamps or a displayed message by using causal rules and time data. The failure messages are created by the Built-In Test Equipment (BITE). For every failure message a fault item is generated. The CMS correlates new failure messages and effects to open fault items. This correlation is based on the timestamp of an occurring fault, the class of the failure message and the ATA (Air Transport Association) chapter of the accused LRUs and the ATA chapter of the emitting LRU. The ATA chapter identifies the aircraft system a LRU belongs to and is defined by the Air Transport Association, an umbrella organization for American airlines. If a failure message can not be correlated to an existing fault item, a new fault item is created. For each fault item, a root cause is determined. But the rules can determine several different root causes for a given fault item. This way a fault item can have up to ten root causes and for every root cause up to three LRUs can be accused. In the worst case thirty possible cases have to be considered when repairing the fault. The maintenance technician uses his experience to filter the list of root causes and LRUs to identify the most probable starting point. Because not every technician has the same experience, the use of CBR to store and share experience and enhance the diagnosis with this experience could be a viable way to improve diagnosis and maintenance with a decision support system. Several use cases of the decision support system were discussed in context of the OMAHA project: application in daily operations in the Operations Control Center for diagnosis during flight, application in the Maintenance Control Center to support unscheduled maintenance tasks, and application at Line Maintenance at the aircraft to support the maintenance technician directly. Two use cases were chosen to be addressed with our decision support system.

Use case 1 is the Airbus Maintenance Control Center. Before landing aircrafts send a PFR and their operational parameters during the flight to the Maintenance Control Center (MCC). The operators in the MCC check the PFR and instruct the maintenance technicians for their preparations. In the MCC the decision support system can be used to retrieve similar cases for every item on the PFR. This way it will be possible to confirm the solution from the rule-based diagnosis, change the ranking of the possible root causes or even provide a list of corrective LRUs, if the accused LRUs from the PFR differ from the list in the cases. If the MCC of an airline has a request for an advice, the MCC of Airbus can provide the results of the decision support system to the respective airline. Figure 1 illustrates the use case.

Use case 2 is the Airbus Customer Service Support. An airline calls the Customer Service and requests support for an fault, that could not be resolved with the knowledge of the airline alone. The Customer Service operator request the data sources for solutions from successfully solved faults and provides several possible solutions to the airline. The solutions from the data sources are not ranked. Almost 70 percent of the requests from airlines can be solved by the Customer service within 30 to 60 min. The other 30 percent have to be passed to the Airbus engineering and require more effort in time and resources to solve. The decision support system will be added to the data sources of the Customer service operator and provides similar cases to the given fault. This way the Customer service

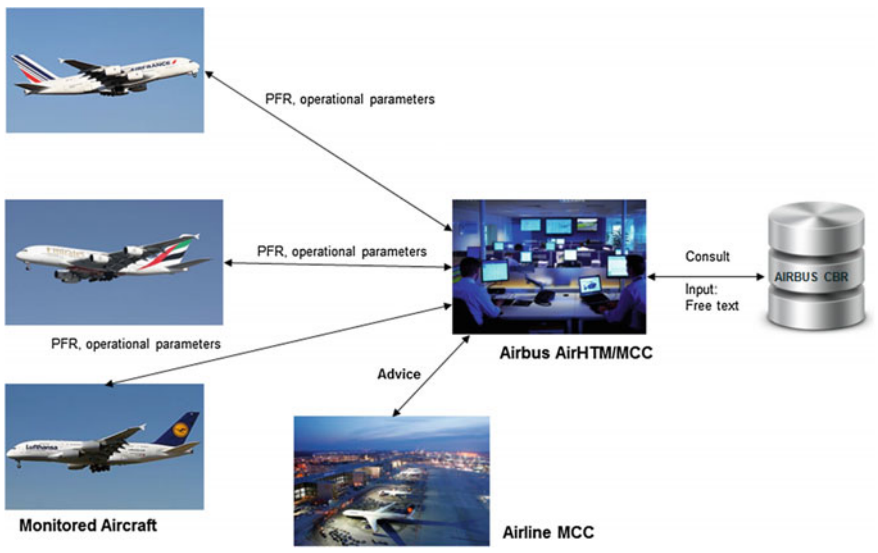


Fig. 1 Use case maintenance control center at airbus

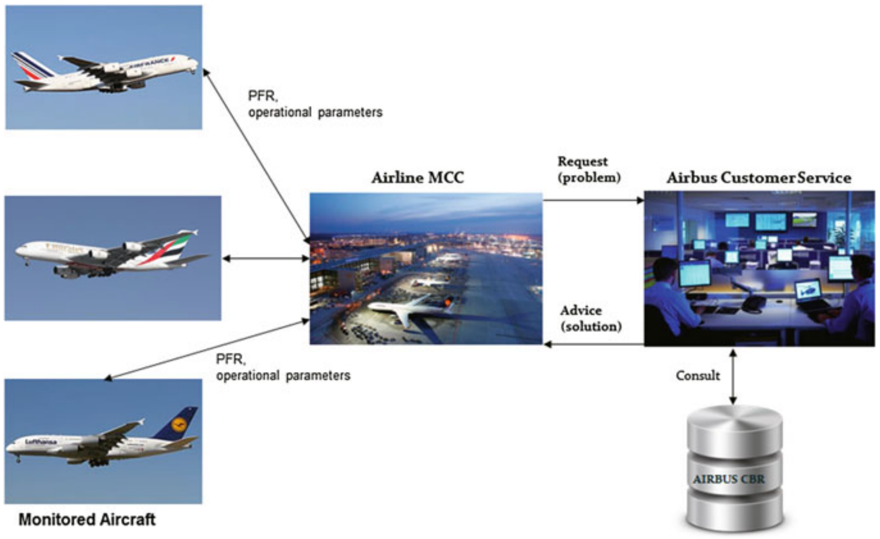


Fig. 2 Use case customer support service at airbus

operator will get a list of ranked cases with successful solutions to similar faults. The goal of this use case is to enable the operator to solve requests from airlines faster and solve faults, that have to be passed to the Airbus engineering in the past (Fig. 2).

In both use cases, the decision support system will not replace the existing rule-based diagnosis, but will enhance it with experience knowledge to identify a root cause and the responsible LRU quicker and more precisely.

3.2 OMAHA Project

The OMAHA project is supported by the Federal Ministry of Economy and Technology in the context of the fifth civilian aeronautics research program [20]. The high-level goal of the OMAHA project is to develop an integrated overall architecture for health management of civilian aircraft. The project covers several topics like diagnosis and prognosis of flight control systems, innovative maintenance concepts and effective methods of data processing and transmission. A special challenge of the OMAHA project is to outreach the aircraft and its subsystems and integrating systems and processes in the ground segment like manufacturers, maintenance facilities, and service partners. Several enterprises and academic and industrial research institutes take part in the OMAHA project: the aircraft manufacturer Airbus (Airbus Operations, Airbus Defense and Space, Airbus Group Innovations), the system and equipment manufacturers Diehl Aerospace and Nord-Micro, the aviation software solutions provider Linova and IT service provider Lufthansa Systems as well as the German Research Center for Artificial Intelligence and the German Center for Aviation and Space. In addition, several universities are included as subcontractors.

The OMAHA project has several different sub-projects. Our work focuses on a sub-project to develop a cross-system integrated system health monitoring (ISHM). The main goal is to improve the existing diagnostic approach with a MAS with several case-based agents to integrate experience into the diagnostic process and provide more precise diagnoses and maintenance suggestions.

3.3 Multi-agent System for Decision Support

Our multi-agent system is based on the SEASALT architecture. The SEASALT (Shared Experience using an Agent-based System Architecture Layout) architecture is a domain-independent architecture for extracting, analyzing, sharing, and providing experiences. The architecture is based on the Collaborative Multi-Expert-System approach [21, 22] and combines several software engineering and artificial intelligence technologies to identify relevant information, process the experience and provide them via an user interface. The knowledge modularization allows the compilation of comprehensive solutions and offers the ability of reusing partial case information in form of snippets [23].

The SEASALT architecture consists of five components: the *knowledge sources*, the *knowledge formalization*, the *knowledge provision*, the *knowledge representation*, and the *individualized knowledge*. The *knowledge sources* component is responsible for extracting knowledge from external knowledge sources like databases or web pages and especially Web 2.0 platforms, like forums and social media platforms. The *knowledge formalization* component is responsible for formalizing the extracted knowledge into a modular, structural representation. The *knowledge provision* component contains the so called Knowledge Line. The basic idea is a modularization of knowledge analogous to the modularization of software in product lines. The modularization is done among the individual topics that are represented within the knowledge domain. If CBR system is used as knowledge source, the SEASALT architecture provides a Case Factory for the individual knowledge maintenance [23, 24]. The *knowledge representation* component contains the underlying knowledge models of the different agents and knowledge sources. The synchronization and matching of the individualized knowledge models improves the knowledge maintenance and the interoperability between the components. The *individualized knowledge* component contains the web-based user interfaces to enter a query and present the solution to the user [23]. For more details on the SEASALT architecture see [23] and for details about the Case Factory approach see [25, 26].

Our multi-agent decision support system is an instantiation of the SEASALT architecture. The central component of our system is the *knowledge provision*, where the Knowledge Line is located. The Knowledge Line is responsible for retrieving similar problems for a given fault situation and providing a diagnosis and the performed maintenance actions. Therefore, several software agents are used to receive a query and retrieve a solution. A communication agent receives the input from a user and sends it to the coordination agent. The coordination agent is responsible for distributing the query to the relevant topic agents. Each topic agent has access to a CBR system, performs a retrieval, and delivers the found cases to the coordination agent. The knowledge is distributed among the CBR systems of the topic agents and is divided using aircraft types (e.g., A320, A350, or A380) and aircraft systems (cabin, ventilation control, hydraulic) as distinctions. Each system is identified by the so-called ATA chapter, a number of four or six digits. This way, one CBR system contains cases for A320 cabin faults, another CBR system contains cases for A380 ventilation control faults.

The approach of having individual agents for each aircraft type and ATA chapter is based on the idea to distribute the knowledge among CBR systems to decrease the modeling, retrieval, and maintenance effort for each CBR system. The coordination agent decides which topic agents are required to find a solution for the query. This decision is based on the aircraft type and the ATA chapter. Nevertheless, the cases in the other case bases may contain useful information as well. Especially when the primary topic agents cannot provide a sufficient solution. Therefore the query can be distributed to the other topic agents as well, because faults and their maintenance recommendations may be similar in different aircraft types.

The last agent in the *knowledge provision* is the so-called query analyzer agent. This agent is responsible for analyzing the query and identifying new concepts, which are not part of the vocabulary of the CBR systems and identifying new similarity information. If any new concepts or similarity information are found, a maintenance request is sent to a Case Factory. The Case Factory derives appropriate maintenance actions and notifies a knowledge engineer about the changes. To analyze the query and performing the derived changes, parts of a workflow for knowledge transformation are used. These tasks combine natural language processing techniques and CBR mechanisms to identify new knowledge and transform it to be used by the CBR systems.

The user interface is located in the *individualized knowledge* component. It is a web interface, which provides options to send a query, perform a retrieval, present the solutions, enter new cases, and browse the case bases. Another interface in the component links to a data warehouse, where fault information is stored, which can be used as input for our decision support system. Via the interface, a query can be received and the solutions sent back to the data warehouse.

The *knowledge formalization* component transforms structured, semi-structured, and unstructured data into a modular, structural knowledge representation used by all CBR systems. This way the knowledge is represented in the same way all over the MAS. The complete version of the workflow for knowledge transformation and formalization is used by a so-called case base input analyzer. The workflow consists of several steps: preprocessing of the input data, collocation extraction, keyword extraction, synonym identification, vocabulary extension, association rule mining, clustering and case generation and sensitivity analysis. The individual steps are described in more detail in Sect. 3.4.

In the *knowledge sources* component a collector agent is responsible for finding new data in the data warehouse, via web services or in existing knowledge sources from Airbus. New data can be new configurations or operational parameters, new synonyms or hypernyms, or complete new cases.

The *knowledge representation* component contains the generated vocabulary, similarity measures and taxonomies, completion rules, and constraints provided for all agents and CBR systems (Fig. 3).

3.4 FEATURE-TAK

This section describes FEATURE-TAK, an agent-based **F**ramework for **E**xtraction, **A**nalysis, and **T**ransformation of **U**nstructe**RE**d **T**extual **A**ircraft **K**nowledge. We will describe the idea, the agent-based architecture and the individual tasks of the framework. Details of the framework can be found in [18] and we will focus in this chapter on the changes and additions to the framework. The description will be supported with a running example to show the input data and results of the individual tasks.

Airbus has collected a lot of data in the last years about maintenance problems and their solutions. These data sets are based on Airbus internal problems or on problems with Airbus aircraft in use by different airlines. More than 500.000 datasets are stored in Airbus databases and they contain information about aircraft type and model, occurred problems, accused LRUs, preferred maintenance actions, executed maintenance actions, comments, and documentation references. These information

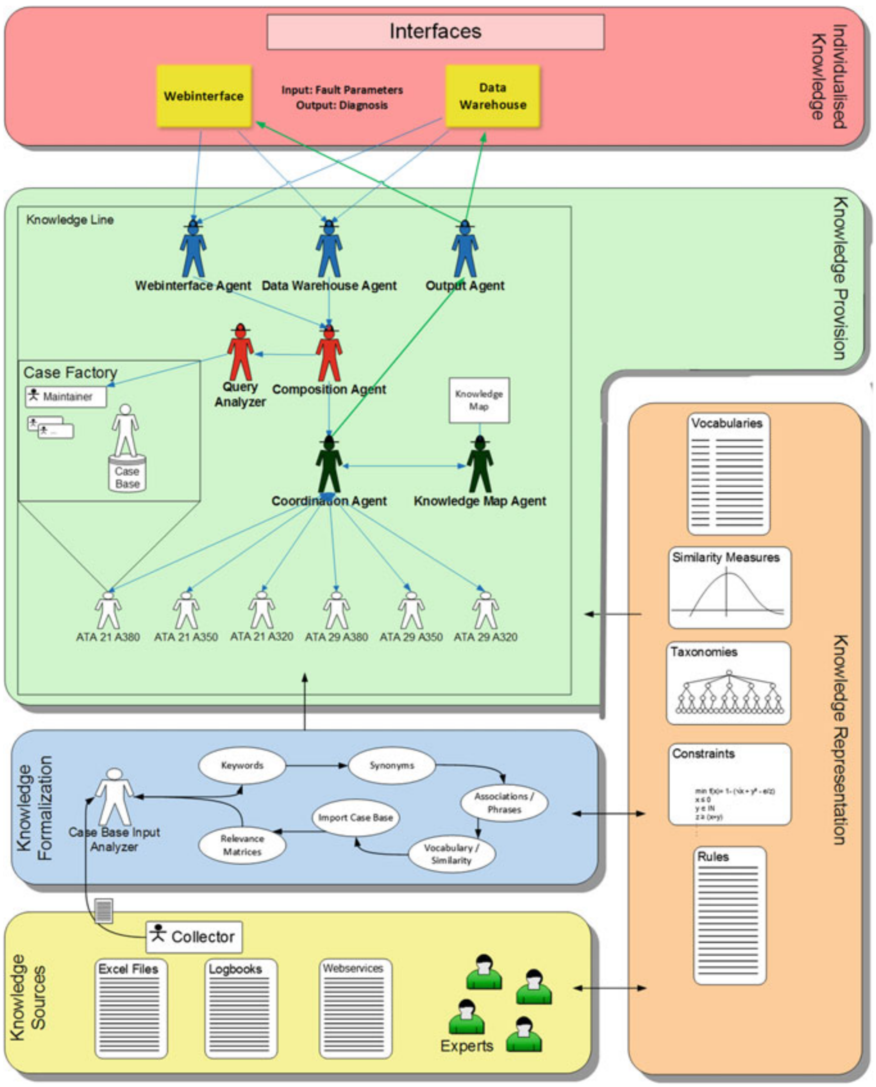


Fig. 3 Instantiation of the SEASALT architecture within the OMAHA project

can be accessed with two tools. One is called One-Search-For-All (OSFA) and the other World-InService-Experience (WISE). Both tools support full text search and a simple similarity based search based on synonyms, but they do not rank the results. Therefore, it can be a time consuming process to find helpful datasets, because the most helpful information is not always shown on the first three pages of the result list.

To use this information within our case-based diagnosis system, they have to be analyzed to find relevant pieces of information to be transformed into knowledge for CBR systems. The data sets from Airbus contain different data structures. Technical information can mostly be found in attribute-value pairs, while logbook entries, maintenance, and feedback are stored in form of free text articles. Based on manual data analysis of several hundred data sets, we choose a structural approach for our CBR systems. Information like fault codes, aircraft type and model, ATA chapter and fault emitter are important information and can easily be used within a structural approach. During the development process it turned out, that the information in the free texts are very important to identify a problem. Therefore, we have to use the structured information as well as the free text. To transform the relevant information in the free texts into useful knowledge for our structural CBR system, we had to adapt and combine techniques from NLP and CBR. The idea is to develop a framework to combine several techniques and automatize the knowledge transformation. This framework could be used for knowledge acquisition and maintenance for CBR system in the development phase or for existing CBR systems. It could be embedded into the knowledge formalization layer of the SEASALT architecture or used as a standalone application.

The framework consists of five components: data layer, agent layer, CBR layer, NLP layer and interface layer. The data layer is responsible for storing the raw data and the processed data for each task. In addition, domain specific information like abbreviations and technical phrases are stored in this layer to be accessible for the other components. The agent layer contains several software agents. For every task an individual agent is responsible. All task agents communicate with a central supervising agent. This supervising agent coordinates the workflow. Some tasks could be processed in parallel, while others could be deactivated by the users configuration. For visualization and communication purposes for the user, this layer also contains an interface agent. For each task an agent is spawned when starting the framework, but additional agents can be spawned to support the initial agents while processing huge data sets or handling parallel queries. The NLP layer contains algorithms and methods like part of speech tagging, lemmatization, abbreviation replacement and association rule mining. These algorithms are used by the agents to execute their assigned tasks. The algorithms could either be third party libraries or own implementations. The fourth layer is the CBR layer and is responsible for the communication with a CBR tool like myCBR or jColibri. It contains methods to add keywords to the vocabulary, extend similarity measures and generate cases from the input data sets. The last layer contains the graphical user interface of the framework. This user interface can be used to configure the framework, select input data, and start the workflow. In addition, the user interface presents the results of each task to the user and shows the status of the software agents.

In the following we will give an short overview of the framework tasks and describe the additions and changes that were made since the last publication [18]. These tasks and their interaction are defined based on the existing input data and the required data structure for our CBR systems. Based on our initial idea, the experience from the input data analysis and the feedback from test with Airbus and Lufthansa experts, we

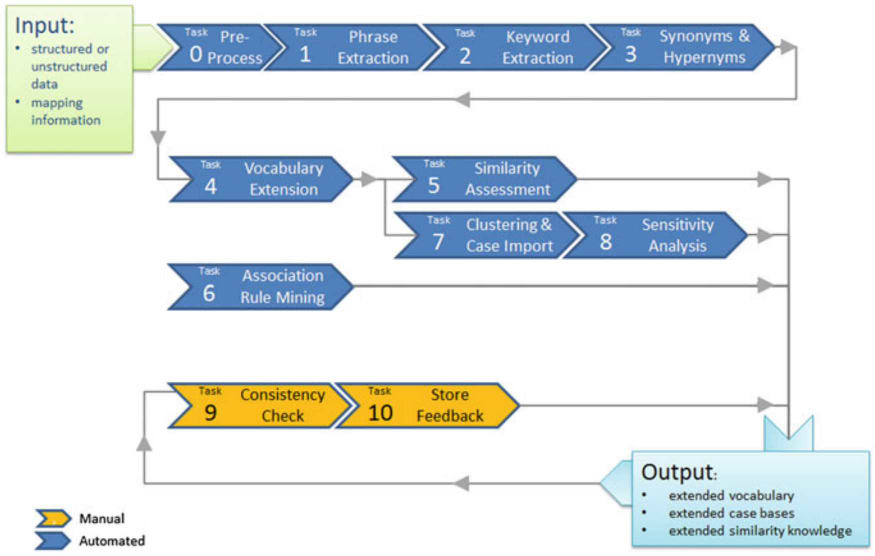

Input:
structured or unstructured data
mapping information
Task 0 Pre-Process
Task 1 Phrase Extraction
Task 2 Keyword Extraction
Task 3 Synonyms & Hypernyms
Task 4 Vocabulary Extension
Task 5 Similarity Assessment
Task 7 Clustering & Case Import
Task 8 Sensitivity Analysis
Task 6 Association Rule Mining
Task 9 Consistency Check
Task 10 Store Feedback
Output:
extended vocabulary
extended case bases
extended similarity knowledge
Manual
Automated

Fig. 4 Tasks processed by FEATURE-TAK and knowledge engineer

had to regroup and extend the tasks and define a new task for preprocessing the data sets. As input for the framework a data set with free text components, for example a CSV file, or a pure free text document like PDF is possible. In addition to the data sets, a file with mapping information, an abbreviations file, and files with domain specific white and black lists are used. The data sets are first transformed into an internal representation of our case structure based on the information in the mapping file. It is not required to have information for every attribute in the data set or to use all information in the data set. The complete case structure for our use case consists of 68 attributes with different data types and value ranges and the mapping process adapts dynamically to the input information. The complete workflow with all tasks and possible parallelization is show in Fig. 4.

As an example for the tasks we assume the following text as input data: “after to the leftside engine reported an err that lead to immediate ret to the airport.”

Preprocessing

The preprocessing task is a new task, which is responsible for performing all function on the input data that are required to do the following tasks. It is called task zero. This tasks covers currently part-of-speech (POS) tagging and abbreviation identification. POS tagging is the first step of this task to identify the word classes in a given text and enabling the abbreviation identification and all later tasks. The abbreviation identification is a required, because the domain specific abbreviation contain some abbreviations that are identical with words in common english language, like “and”, “of”, “at”, “in” or “to”. For example is “of” the short form for “overfrequency” and “to” for “takeoff”. It is important to identify whether this words are abbreviations or not, because a wrong replacement could lead to a different sense of a sentence and therefore different extraction results. Therefore, a probability is computed to decide whether a word is an abbreviation or a preposition or stopword. The basic probability for a word is 0.5. Based on the surrounding words and the determined type dependency of the POS tagging, the probability is increased or decreased. If the probability is greater than 0.6 it is treated as a preposition, if the probability is smaller than 0.4 it is treated as an abbreviation. In our example three times the word "to" exists. Only the first occurrence is an abbreviation, the other words are prepositions. The probability for the prepositions is 0.7 and the probability for the abbreviation is 0.2. The result of this task is a sentence with an replaced first "to" with "takeoff" and annotated words.

Collocation Extraction

The first task is the identification and extraction of phrases from the free text components of the input data. The idea is to find recurring combinations of words based on standard english grammar and domain-specific terms. This task has two steps: multi-word abbreviation identification and phrase extraction. First, multi-word abbreviations are identified, because the longform of these abbreviations counts as phrases too. The second step is to identify phrases based on the tagging information and the word position in a sentence.

Unfortunately, many free text components do not follow a coherent grammatical structure, and therefore are ineligible for POS tagging. To solve this problem, the text corpus may be manually inspected for patterns. In the OMAHA project a dataset of fault description was provided by Airbus, and patterns were manually created with the help of regular expressions to fit the language and expressions used by the experts. With the help of these patterns keywords and phrases are extracted.

To give an example of what the patterns might look like, we look into fault descriptions of up to three words and try to distinguish between a system and its fault status. Let's say we have a pre-existing list of statuses, *STATUS* is a term from that list, and *w* is a word, then:

• *w STATUS*	*w* is the system, e.g. "apu shutdown"
• noun phrase *STATUS*	the noun phrase is the system, e.g. "selcal lights flashing"
• *w* is *STATUS*	*w* is the system, e.g. "tcas is inop"

In this example the keywords and phrases which were extracted are "apu", "selcal lights", and "tcas", and labeled as $system$. Other labels include the $location$ of the fault, and $time$, describing when the fault happened. In a similar vein, patterns were written for fault descriptions with 4 words, and for 5–6 words. Longer description tended to be less coherent and so no patterns were produced for them. The result of this tasks is a modified free text, reduced by multi-word abbreviations and found phrases. Based on the example no multi-word abbreviations are found, but a phrase was identified: "leftside engine".

Keyword Extraction

The second task of the framework is the extraction of keywords from the remaining text, and consists of three steps: stopword elimination, lemmatization, and single-word abbreviation replacement. As input for this task, the modified text from task one is used. The stopword elimination is based on common english and a white list of words that should not be eliminated. In the second step, for all words the lemmata are determined. The third step identifies abbreviations in the remaining words and replaces them with their longform. The result of the complete task is a list of keywords, reduced to their base form. Based on the example the following keywords were extracted: after, report, error, lead, immediate, return, airport.

Synonyms and Hypernyms

The third task is responsible for identifying synonyms and hypernyms for the extracted keywords and phrases. Therefore, the input for this task is a list of phrases from the first task and list of keywords from the second task. One challenge in this task, is to consider the context and word sense of a keyword to identify the right synonyms. Therefore, we are using the part of speech information and a blacklist of words, that should not be used as synonyms. The second step is to identify the hypernyms for all keywords. The result of this task is a list of phrases and keywords and their synonyms and hypernyms. For our example the following synonyms could be identified:

- after: behind, afterwards, later
- report: inform, notify
- error: failure, fault, faux pas
- airport: terminal, airfield

The synonym faux pas for error is filtered by the blacklist, because it is an inappropriate synonym.

Vocabulary Extension

This task consists of adding the extracted keywords, phrases, synonyms, and hypernyms to the vocabulary of the CBR systems. The first step is to remove duplicate words and phrases to avoid redundant knowledge. The second step is to check the list of keywords against the list of phrases to identify keywords which occur as phrases. We want to slow down the growth of the vocabulary and therefore we identify keywords that only occur as part of a collocation. These keywords are not added to the vocabulary. If a keyword occurs without the context of a collocation, it will be added.

Similarity Assessment

After extracting the keywords and phrases and extending the vocabulary in the previous tasks, we turn to determine how similar the items in the vocabulary are to each other. This is important for the retrieval task, as a global similarity between cases needs to be established, and to this end a proper local similarity measure is needed.

One basic but effective similarity measure is taxonomies. They allow to model the semantic relationship between keywords in the vocabulary, while determining their similarity. One way to obtain the taxonomies is manually with experts, as they provide initial information about major concepts in the dataset. Once the basic ideas are there, another way to create taxonomies is with the help of synonyms and hypernyms. As mentioned before, synonyms and hypernyms are extracted from the existing keywords and added to the vocabulary, while extending existing taxonomies or generating new ones.

Unfortunately, when processing natural language text written by different contributors, there are many concepts and terms that appear in very few cases. This creates a long tail of infrequent but important terms that experts cannot possibly manually model in taxonomies, and have not been found to be synonyms or hypernyms of other concepts. In order to include these terms we regard the relationship between concepts and cases as a bipartite social network, where a concept is linked to a cluster of cases (as describe in 3.4) if it appeared in at least one case in that cluster. With the help of social network analysis (SNA) methods, namely weighted one-mode projection (OMP) a similarity between the concepts can be estimated [27]. This is done by performing the weighted OMP over the concepts, calculating the weight between them, and then using them as the local similarity value.

This method results in an asymmetrical similarity function that maps each two concepts to the similarity value between them. It is fully automated and takes into account the influence of each concept on each cluster, and the cooccurrence of the concepts.

Association Rule Mining

In the aircraft domain many causal dependencies between systems, status, context and functions exist. Association rule mining (ARM) can help identify this dependencies in an automated way to avoid the high effort from manually analyzing the data sets. This task is used to analyze the keywords and phrases and find associations between the occurrence of these words within a data set as well as across data sets. We try to identify reoccurring associations to determine completion rules for our CBR systems to enrich the query. An association between keywords or phrases exists, when the combined occurrence exceeds a given threshold. For example, a combination between two keywords that occurs in more than 80% of all analyzed documents, may be used as a completion rule with an appropriate certainty factor. Completions rules could lead to more information in a query and therefore additional cases could be retrieved. Because the ARM algorithms are very time consuming with huge datasets, we decided to add a benchmark to this task. The benchmark processes either the input data sets or all cases in all case bases and determines the interval of possible support and confidence values. This way we could display to the user for which support and confidence values rules could be extracted. Based on the results, the user can decide to perform the ARM with the Apriori [28] or the FPGrowth [29] algorithm, if the interval contains the desired support and confidence values. Another addition to this task is to consider the lift value for association rules. The lift determines the usefulness of a rule and is the ratio between the confidence value of the rule and the expected confidence value [30]. Using the lift and determining an threshold reduces the generated rules and ensures a minimum information gain.

Case Generation and Clustering

This task is responsible for generating a case from each input data set and storing it in a case base. To avoid a large case base with hundreds of thousands of cases, we cluster the incoming cases and distribute them to several smaller case bases. Generating an abstract case for each case base, a given query can be compared to the abstract cases and this way a preselection of the required case bases is possible. For each aircraft type (A320, A330, A340, etc.) a set of case bases will be created and each set will be separated by the ATA chapter.

Sensitivity Analysis

The idea behind the sensitivity analysis is that different attributes of a case are differently important for the global similarity in each case base. For example, the location attribute of a fault related to defected seat belts may not be as important as it would be when the fault is related to eliminating bad odors. As mentioned before, there are several case bases divided by aircraft type and ATA chapter. During a retrieval the topic agents are comparing the query to the different case bases, and each case base has its own set of attribute weights.

There are two main stages for the sensitivity analysis, namely the statistical analysis and the learning stage. The statistical analysis phase finds the diversity of the attribute values, and their impact on the attribute in each case base. The results of this phase are then used as a starting point for the optimization performed in the learning phase. Here supervised learning is used. For each retrieval only retrieved cases with a similarity value above a certain threshold are observed. A retrieved case is relevant if it was extracted from the correct case base (i.e. it was diagnosed correctly), and is considered a false positive if its similarity was above the threshold but it came from the wrong case base. The error in this situation would then be $error = sim - threshold$. The weights belonging to the irrelevant case base are updated by their relative contribution to the error. This optimization is performed in the course of a predefined number of epochs for computational complexity reasons and for each one the F-measure is calculated. The values that produced the highest F-measure score are then used.

The output of the sensitivity analysis is a relevance matrix where each row represents a different case base, the columns represent the attributes, and the cell values are the weight of the given attribute in the given case base. A schema of the matrix can be seen in Table 1. These weights are then used when the local similarities are amalgamated to find global similarities [19].

Table 1 The relevance matrix. The rows represent case bases, while the columns represent attributes

	a_1	$\dots$	a_m
CB_1	w_{11}	$\dots$	w_{1m}
$\vdots$	$\vdots$	$\ddots$	$\vdots$
CB_s	w_{s1}	$\dots$	w_{sm}

3.5 Knowledge Modeling

Based on our initial data analysis at the beginning of the OMAHA project, we decided to use the structured CBR approach and represent the knowledge as attribute-value pairs. Much knowledge is stored in databases or CSV files and has a unique column-value correlation, and it can easily be transformed into attribute-value pairs. However, during the progress of the project knowledge in form of free text became more and more important, because these free texts contain many relevant experience. A pure textual CBR approach is not viable, because the structured information is also important for a diagnosis. Therefore an approach was required that considers the structured information as well as the free texts. Because the knowledge from the cases in our CBR systems should be stored in the data warehouse as well, we decided to stay at the structured CBR approach and try to transform the information of the free text into a structured representation. To transform the unstructured knowledge we use the FEATURE-TAK framework.

The knowledge modeling was done manually and automated. The modeling of the case structure, the initial vocabulary and similarity measures were modeled manually with the help of experts from Airbus and Lufthansa. After the initial implementation of FEATURE-TAK the knowledge modeling was also done automated, but has always manually tested and enhanced, if necessary. As initial data sources several databases, document collections, articles and reports were given. We used Service Information Letters (SIL), In-Service Reports (ISR), PFRs, articles from OSFA and WISE, and operational parameters. SIL contain information about exception handling during the maintenance process. ISR are reports from the Airbus customer service and contain help requests from airlines with the corresponding solution. PFR contain failure items generated during the flight of an aircraft. The articles from OSFA and WISE contain also information about maintenance incidents, but usually have additional information and references to other documents. The operational parameters contain outside temperature during different flight phases, outside air pressure, flight height, and start and destination airport.

Several challenges were faced during the knowledge modeling process. The first challenge was to identify the relevant information that could be used in the CBR systems. We had to analyze the data sources and discuss with the experts which information could be used to describe the problem and which information are required as the corresponding solution. Another challenge was to model initial similarity measures. While aircrafts like the A380 and A350 have some similar systems, they have also great differences in engines, cabin systems and cockpits. In addition they have a different maintenance concept, which leads to different failure messages or effects for the same fault and the ATA chapter is not fully compatible between aircrafts. The same system in A380 and A350 are dedicated to different ATA chapters. Therefore, modeling similarity measures cross aircraft compatible is very challenging. Also challenging was the problem, that one data set or document can contain information about more than one fault or the same fault was described in different ways. Especially the logbook information from pilots or the cabin crew have to be analyzed carefully to identify the faults and additional relevant information.

From the different data sources we derived a case structure with 71 attributes. These attributes are distributed among problem description, solution, quality information and additional information. The problem description consists of 21 attributes. The knowledge modeling process focused on these attributes, because the problem description is relevant to identify similar cases and therefore possible similar solutions. In the problem description the following information are modeled:

- aircraft type: type of an aircraft, e.g. A380, A350, A320
- aircraft model: specific model of an aircraft, e.g. 320–200, 350–1041
- ATA chapter: ATA chapter of the accused LRUs, e.g. 230056
- emitter: system that has emitted the failure message, e.g. air conditioning
- faultcode: faultcode of the fault, e.g. 4411F170
- engine type: type of engine, e.g. cfm56-5a1
- delay duration: delay duration that was caused by the fault in minutes
- displayed message:message displayed in the cockpit or cabin
- fault system: systems that are mentioned in the problem description
- fault location: locations that are mentioned in the problem description
- fault status: systems status that are mentioned in the problem description
- fault function: functions of the systems that are mentioned in the problem description
- fault time amount:time amount mentioned in the problem description
- fault time unit: time unit, e.g. hour, minutes
- fault time information: additional time information, e.g. after start, during takeoff
- affected LRU 1–3: LRU accused by the BITE failure message, e.g. handset, vent, relay
- affected LRU 1–3 PN: partnumber of the accused LRUs, for direct identification

For each affected LRU 1–3 and the corresponding part number an own attribute is modeled in the case structure. The information extracted from free texts by FEATURE-TAK are stored in the attributes: fault system, fault status, fault function, fault location, fault time amount, fault time unit and fault time information. Distributing the information from free text over several attributes reduce the modeling and maintenance effort for each attribute, especially the taxonomy-based similarity measures. Figure 5 shows sample values for these attributes.

The solution contains 8 attributes with information about performed work, recommendation, corrective LRU 1–3, comments, documentation references, and the complete problem description. For the knowledge modeling only the attributes for corrective LRUs are considered. The other attributes contain only free text directly from the data sources. The complete free text problem description is also modeled as part of the solution to allow the users of the system a comparison between the full text problem description of a retrieved case and the query.

As quality information the amount of correct and false retrievals of a case are stored. This information is used to adapt the similarity of the retrieved cases to the query. If the case was successful in its assistance in previous requests, the similarity

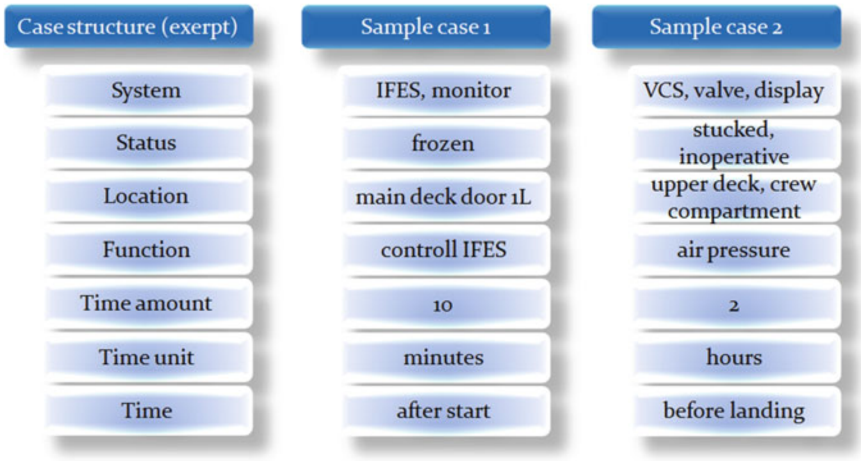


Fig. 5 Excerpt from the case structure with sample values

value may shift upwards by up to $\frac{1}{3}(1 - sim)$, but if it was more unhelpful than helpful its similarity value will decrease by up to $\frac{1}{3}sim$.

The other 40 attributes contain additional information that may be useful to the operators or technicians or are meta-information like case id, creation date, departure airport, flight date, delay times and maintenance time, flight phase, and special incidents. The discussion with Airbus and Lufthansa experts whether several of these attributes should be integrated into the problem description is not finished. Attributes that contain information about operational parameters or special incidents could be part of the problem description to have a more precise retrieval.

The similarity measures for the problem description are based on taxonomies and similarity matrices. Only the attribute delay duration has an numeric interval value range and its similarity measure is based on a symmetric polynomial function. The other attributes have a so-called symbolic data type. For each attribute a set of allowed values is defined. The number of values in these sets is between 18 values for the aircraft type and more than 4000 values for the fault system attribute. The similarity measures for the problem description attributes are modeled as taxonomies, because the values have an hierarchical structure, that could be represented by the taxonomies. In addition, it is less effort to model similarity values for 5–40 layers in a taxonomy, than for an 4000×4000 matrix. For the ATA chapter attribute several different similarity measures are required to compare systems of different aircraft types. Therefore, a similarity function for each pair of comparable aircraft types is modeled. These similarity functions only contain similarity values between ATA chapters that differ in the chosen aircraft types. Otherwise the standard similarity measure is used.

Overall the vocabulary contains more than 40.000 values distributed among the attributes, most of them extracted by FEATURE-TAK.

4 Implementation and Test Runs

The current version of the decision support system is implemented in two parts: a diagnosis system with several CBR systems and a standalone version of FEATURE-TAK. An interface was implemented to load the results from the workflow into the decision support system. The stand-alone version is used for extensive testing by our project partners and therefore not fully integrated into the decision support system. The automated tasks of FEATURE-TAK itself are fully implemented, but the single tasks should be improved. The MAS contains eleven software agents and seven CBR systems. The MAS and the workflow are implemented with JADE [31] and the CBR systems with myCBR [32]. Over all CBR systems we currently have more than 500 cases, some based on manual input, but the most generated with FEATRUE-TAK based on given data sets.

The demonstrator for the decision support systems contains several functions: retrieval of cases, adding new cases, editing existing case, browsing all case bases and storing feedback for cases. For each CBR system a topic agent is implemented to perform the retrieval. The web-based gui can be used to enter a query with structured and unstructured text. The given problem description is then mapped to the case structure. If the query contains words not known by the vocabulary, the query analyzer logs this words. These words can be added manually to the vocabulary by a knowledge engineer. A query is primarily send to the CBR system with the corresponding aircraft type. If the retrieval does not deliver similar cases or not the requested amount of similar cases, the other CBR systems are requested, too. The retrieved cases of one or more CBR systems are displayed to the user. The user can give feedback for each case, if it was helpful or not. These information are stored for each case and are used by future retrievals to rank a case up or down. Helpful cases get a bonus to similarity, while the similarity for unhelpful case is reduced.

For the implementation of FEATURE-TAK several third party tools and libraries are used, as well as own implementations of algorithms. The framework is also agent-based and could be integrated into the demonstrator. This integration process is not finished. In its current implementation, the framework contains eleven agents: supervising agent, gui agent, preprocessing agent, collocation agent, keyword agent, synonym agent, vocabulary agent, similarity agent, ARM agent, cluster agent, sensitivity agent. The gui agent controls the user interface, gets the input file and is responsible for displaying the results of each tasks as well as the overall results of the workflow. In addition, status information about the workflow are shown. The supervising agent is the central agent of the framework. He receives the input file from the gui agent and routes the communication between the task agents. The preprocessing agent is implemented using the POS tagger from the Stanford Core NLP library [33] and an own java implementation for the algorithm to identify abbreviations. The collocation agent also uses the Stanford Core NLP library and an own implementation for the pattern recognition algorithm and the abbreviation replacement. The keyword agent is implemented using Apache Lucene [34] and the Stanford Core NLP library for stopword elimination and lemmatization, while the abbreviation replacement is also an own implementation. For the abbreviation replacement in both agents, a list of domain-specific abbreviations is used that contains 4937 abbreviations. The synonym agent has access to a Wordnet [35] instance with common english synonyms and hypernyms. In addition, the agent uses white and blacklists to reduce or extend the list of synonyms. The vocabulary agent uses the myCBR API to add the extracted phrases, keywords, synonyms and hypernyms to the vocabulary of the CBR systems. The similarity agent also uses the myCBR API to extend the similarity measures. He adds new concepts to existing taxonomies and is able to create a new taxonomy if necessary. The ARM agent has access to an implementation of the Apriori and the FPGrowth algorithm. In addition, he contains an own implementation of the benchmark. The benchmark determines the support and confidence values which can produce results and displays these results in a scatter plot. The cluster agent generates cases for all input data sets and distributes them among the existing case bases. He uses the myCBR API to generate the cases and the clustering algorithm is an own implementation. The algorithm distributed the cases based on the aircraft type and the ATA chapter. If an additional case base is required, the cluster agent is able to generate a new case base. The last agent is the sensitivity agent. This agent performs a sensitivity analysis on the existing case for all case bases. The relevance matrix is used to adapt the attribute weights for the global similarity function. The algorithm for the sensitivity analysis is an own implementation.

During the development process several test were run on FEATURE-TAK and the diagnosis system. Airbus and Lufthansa experts tested both prototypes with small sets of data (100–200 data sets) to identify required improvements. These tests were part of an argumentative evaluation to show the feasibility of the decision support system and FEATURE-TAK for different project milestones. The evaluation results can be seen in [17, 18].

4.1 Future Evaluation

Currently, the decision support system is integrated into the overall demonstrator of the OMAHA project. This overall demonstrator provides a simulator, which is capable of simulating a complete fleet of aircraft, injecting faults into a specific aircraft and simulate a maintenance planning. The decision support system will be tested with several injected faults and evaluated from Airbus and Lufthansa experts. In addition, the decision support system will be tested by Customer Service operators from Airbus with real data from In-Service reports. This way the performance and benefit of the decision support system will be evaluated. The FEATURE-TAK framework will be tested with a data set of more than 65.000 In-Service reports to evaluate the quality and performance of the single tasks.

5 Summary and Outlook

In this chapter we described out approach for an multi agent decision support system for diagnosis and maintenance in the aircraft domain. We give an overview of the SEASALT instantiation, the diagnosis workflow and our framework FEATURE-TAK for analyzing free texts and extracting knowledge from them. Currently, the diagnosis system and FEATURE-TAK are integrated into a single demonstration system, that will be integrated into the overall demonstrator for evaluation purposes. Additional future work is the implementation of Case Factories for every CBR system for knowledge maintenance and extending the learning capabilities of the system.

References

1. Jackson, T., Austin, J., Fletcher, M., Jessop, M.: Delivering a grid enabled distributed aircraft maintenance environment (DAME). Technical Report, University of York (2003)
2. Feret, M., Glasgow, J.: Combining case-based and model-based reasoning for the diagnosis of complex devices. Appl. Intell. **7**, 57–78 (1997)
3. Corchado, J.M., Tapia, D.I., Bajo, J.: A multi-agent architecture for distributed services and applications. Int. J. Innov. Comput. **8**, 2453–2476 (2012)
4. Zouhair, A., En-Naimi, E.M., Amami, B., Boukachour, H., Person, P., Bertelle, C.: Incremental dynamic case based reasoning and multi-agent systems (idcbr-mas) for intelligent touring system. Int. J. Adv. Res. Comput. Sci. Softw. Eng. **3**, 48–56 (2013)
5. Ceausu, V., Desprès, S.: A semantic case-based reasoning framework for text categorization. In: The Semantic Web, pp. 736–749. Springer, Berlin (2007)
6. Rodrigues, L., Antunes, B., Gomes, P., Santos, A., Barbeira, J., Carvalho, R.: Using textual CBR for e-learning content categorization and retrieval. In: Proceedings of International Conference on Case-Based Reasoning (2007)
7. Weber, R., Aha, D.W., Sandhu, N., Munoz-Abila, H.: A textual case-based reasoning framework for knowledge management applications. In: Proceedings of the 9th German Workshop on Case-Based Reasoning, pp. 244–253. Shaker Verlag (2001)
8. Manning, C.D., Surdeanu, M., Bauer, J., Finkel, J.R., Bethard, S., McClosky, D.: The stanford coreNLP natural language processing toolkit. In: ACL (System Demonstrations), pp. 55–60 (2014)
9. McCandless, M., Hatcher, E., Gospodnetic, O.: Lucene in Action: Covers Apache Lucene 3.0. Manning Publications Co. (2010)
10. Cunningham, H., Maynard, D., Bontcheva, K.: Text processing with gate. Gateway Press CA (2011)
11. Xu, F., Uszkoreit, H., Li, H., Adolphs, P., Cheng, X.: Domain-adaptive relation extraction for the semantic web. In: Towards the Internet of Services: The THESEUS Research Program, pp. 289–297. Springer International Publishing (2014)
12. Petrov, S.: Announcing syntaxnet: The worlds most accurate parser goes open source (2016). https://research.googleblog.com/2016/05/announcing-syntaxnet-worlds-most.html. Google research blog
13. Sizov, G.V., Ozturk, P., Styrak, J.: Acquisition and reuse of reasoning knowledge from textual cases for automated analysis. In: Lecture Notes in Computer Science, pp. 465–479. Springer International Publishing, Berlin (2009)
14. Bach, K., Althoff, K.D., Newo, R., Stahl, A.: A case-based reasoning approach for providing machine diagnosis from service reports. In: Case-Based Reasoning Research and Development. International Conference on Case-Based Reasoning (ICCBR 2011), pp. 363–377 (2011)

15. Reuss, P., Althoff, K.D., Henkel, W., Pfeiffer, M.: Case-based agents within the OMAHA project. In: Case-based Agents. ICCBR Workshop on Case-Based Agents (ICCBR-CBR-14) (2014)
16. Reuss, P., Althoff, K.D., Hundt, A., Henkel, W., Pfeiffer, M.: Multi-agent case-based diagnosis in the aircraft domain. In: Case-based Agents. ICCBR Workshop on Case-based Agents (ICCBR-CBA-15) (2015)
17. Reuss, P., Althoff, K.D., Henkel, W., Pfeiffer, M., Hankel, O., Pick, R.: Semi-automatic knowledge extraction from semi-structured and unstructured data within the OMAHA project. In: Proceedings of the 23rd International Conference on Case-Based Reasoning (2015)
18. Reuss, P., Stram, R., Juckenack, C., Althoff, K.D., Henkel, W., Fischer, D.: Feature-tak - framework for extraction, analysis, and transformation of unstructured textual aircraft knowledge. In: Proceedings of the 25th International Conference on Case-based Reasoning, ICCBR 2016 (2016)
19. Stram, R., Reuss, P., Althoff, K.D., Henkel, W., Fischer, D.: Relevance matrix generation using sensitivity analysis in a case-based reasoning environment. In: Proceedings of the 25th International Conference on Case-Based Reasoning, ICCBR 2016. Springer, Berlin (2016)
20. BMWI: Luftfahrtforschungsprogramms v (2013). http://www.bmwi.de/BMWi/Redaktion/PDF/B/bekanntmachung-luftfahrtforschungsprogramm-5,property=pdf,bereich=bmwi2012,sprache=de,rwb=true.pdf
21. Althoff, K.D.: Collaborative multi-expert-systems. In: Proceedings of the 16th UK Workshop on Case-Based Reasoning (UKCBR-2012), Located at SGAI International Conference on Artificial Intelligence, December 13, Cambridge, United Kingdom, pp. 1–1 (2012)
22. Althoff, K.D., Bach, K., Deutsch, J.O., Hanft, A., Mänz, J., Müller, T., Newo, R., Reichle, M., Schaaf, M., Weis, K.H.: Collaborative multi-expert-systems – realizing knowledge-product-lines with case factories and distributed learning systems. In: Baumeister, J., Seipel, D. (eds.) KESE @ KI 2007. Osnabrück (2007)
23. Bach, K.: Knowledge acquisition for case-based reasoning systems. Ph.D. thesis, University of Hildesheim (2013). Dr. Hut Verlag München
24. Althoff, K.D., Reichle, M., Bach, K., Hanft, A., Newo, R.: Agent based maintenance for modularised case bases in collaborative multi-expert systems. In: Proceedings of AI2007, 12th UK Workshop on Case-Based Reasoning, pp. 7–18 (2007)
25. Reuss, P., Althoff, K.D.: Explanation-aware maintenance of distributed case-based reasoning systems. In: LWA 2013. Learning, Knowledge, Adaptation. Workshop Proceedings, pp. 231–325 (2013)
26. Reuss, P., Althoff, K.D.: Maintenance of distributed case-based reasoning systems in a multi-agent system. In: Seidl, T., Beeks, C., Hassani, M. (eds.) LWA 2014 - Lernen, Wissen, Adaption - Workshop Proceedings. GI-Workshop-Tage “Lernen, Wissen, Adaption” (LWA-2014), September 8–10, Aachen, Germany, pp. 20–30. Aachen University of Technology, Aachen (2014)
27. Zhou, T., Ren, J., Medo, M., Zhang, Y.C.: Bipartite network projection and personal recommendation. Phys. Rev. E **76**, 4 (2007)
28. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: Proceedings of the 20th International Conference on Very Large Data Bases, VLDB ’94, pp. 487–499. Morgan Kaufmann Publishers Inc., San Francisco (1994). http://dl.acm.org/citation.cfm?id=645920.672836
29. Borgelt, C.: An implementation of the FP-growth algorithm. In: Proceedings of the 1st International Workshop on Open Source Data Mining: Frequent Pattern Mining Implementations, OSDM ’05, pp. 1–5. ACM, New York (2005). doi:10.1145/1133905.1133907
30. McNicholas, P.D., Murphy, T.B., O’Regan, M.: Standardising the lift of an association rule. Comput. Stat. Data Anal. **52**(10), 4712–4721 (2008)
31. Bellifemine, F., Caire, G., Greenwood, D.: Developing Multi-agent Systems with JADE. Wiley, New York (2007)
32. Bach, K., Sauer, C., Althoff, K.D., Roth-Berghofer, T.: Knowledge modeling with the open source tool myCBR. In: Nalepa, G.J., Baumeister, J., Kaczor, K. (eds.) Proceedings of the 10th Workshop on Knowledge Engineering and Software Engineering (KESE10). Workshop on

Knowledge Engineering and Software Engineering (KESE-2014), Located at 21st European Conference on Artificial Intelligence, August 19, Prague, Czech Republic. CEUR Workshop Proceedings (2014). http://ceur-ws.org/

33. Manning, C.D., Mihai, S., Bauer, J., Finkel, J., Bethard, S., McClosky, D.: The stanford coreNLP natural language processing toolkit. In: Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations, pp. 55–60 (2014)
34. McCandless, M., Hatcher, E., Gospodnetic, O.: Lucene in Action, 2nd edn. Manning Publications Co., Greenwich (2010)
35. Feinerer, I., Hornik, K.: Wordnet: WordNet Interface (2016). https://CRAN.R-project.org/package=wordnet (R package version 0.1-11)

The Role of Ontologies and Decision Frameworks in Computer-Interpretable Guideline Execution

Paulo Novais, Tiago Oliveira, Ken Satoh and José Neves

Abstract Computer-Interpretable Guidelines (CIGs) are machine readable representations of Clinical Practice Guidelines (CPGs) that serve as the knowledge base in many knowledge-based systems oriented towards clinical decision support. Herein we disclose a comprehensive CIG representation model based on Web Ontology Language (OWL) along with its main components. Additionally, we present results revealing the expressiveness of the model regarding a selected set of CPGs. The CIG model then serves as the basis of an architecture for an execution system that is able to manage incomplete information regarding the state of a patient through Speculative Computation. The architecture allows for the generation of clinical scenarios when there is missing information for clinical parameters.

1 Introduction

Knowledge-based systems draw a clear separation of their control processes, which determine what a system should do, from their global database, which defines what a system knows [1, 2]. This separation can be translated into their basic architecture: a knowledge base where knowledge is represented explicitly rather than in procedural code, and an inference engine that runs that knowledge against information about the state of the world. These systems have a number of desirable features such as: making

P. Novais (✉) · J. Neves
Department of Informatics, Algoritmi Research Centre, University of Minho Braga, Braga, Portugal
e-mail: pjon@di.uminho.pt

J. Neves
e-mail: jneves@di.uminho.pt

T. Oliveira · K. Satoh
National Institute of Informatics, Sokendai University, Tokyo, Japan
e-mail: toliveira@nii.ac.jp

K. Satoh
e-mail: ksatoh@nii.ac.jp

G.J. Nalepa and J. Baumeister (eds.), *Synergies Between Knowledge Engineering and Software Engineering*, Advances in Intelligent Systems and Computing 626, https://doi.org/10.1007/978-3-319-64161-4_10

the information required to solve a decision problem explicit, their maintenance is faster and easier with the separation of domain knowledge and code, and the ability to explain the outputs they produce. In clinical decision support these features play an important role given the need for well-founded and consistent advice [3]. However, one of the most difficult parts in developing knowledge-based Clinical Decision Support Systems (CDSSs) is obtaining the necessary domain knowledge. Experts have a limited schedule, their attention is highly demanded throughout their daily activities, in multiple situations. Since computer-based clinical decision support is not a priority, the task of developing these systems is often overlooked in favour of tasks that are more urgent and impactful in the short term. Despite that, there are vehicles for clinical knowledge that can be used as a support for the development of CDSSs. Such is the case of Clinical Practice Guidelines (CPGs) [4, 5], which are systematically developed statements, based on medical evidence, that provide advice for specific patient states. CPGs cover a wide variety of clinical categories and medical specialities, from diagnosis to management, from family practice to surgery. They aim to promote the standardization of medical practice as a way to prevent deviations responsible for the occurrence of adverse events or medical errors [6]. Additionally, they are an important tool in combating over-expenditure resulting from defensive medicine [7].

The versions of CPGs in machine-readable formats are called Computer-Interpretable Guidelines (CIGs) [8–10]. The advantages of adopting these machine-readable versions over their document counterparts are related with the increased availability of guidelines at the point and moment of care and reduced ambiguity [11]. CIGs also help reduce the vagueness in clinical documents, namely that which stems from the use of fuzzy terms to describe events or recommendations whose reach is difficult to determine [11]. Furthermore, the existence of structured formats allows for the development of automated mechanisms for the interpretation of clinical knowledge, resulting in knowledge-based systems that help physicians make better decisions.

Research in CIG systems started in the late 1980's and took off in the late 1990's and early 2000's [8]. Currently, there are a number of CIG models focused on different aspects of CPG representation. These aspects are related with the basic requirements for building CIG models, namely the representation of [12]:

- Workflows of recommendations, enabling the definition of sequences of recommendations, alternative recommendations, and parallel recommendations;
- Conditions about the state of the patient that restrain the application of clinical recommendations;
- Decision points for the inference of new patient states; and
- Temporal restrictions such as durations, waiting times and periodicities.

When developing a CIG model and corresponding execution engine, these are aspects that must be taken into account. They establish the foundations for the development of higher level functions in CIG execution engines [13]. One of such high level functions is the management of uncertainty, which is a pervasive problem in health care. There are multiple meanings and varieties of uncertainty in health care, which are not often distinguished [14–17]. According to [17], where a taxonomy for uncertainty in health care is proposed, this concept can be defined as the perception of not having knowledge about some aspect of the real world. The form it takes depends on many factors such as the source of uncertainty and how it manifests. The kind of uncertainty addressed throughout this chapter falls within the category of incomplete information, discussed in [14], and it can be described as the lack of knowledge about the correct values for the parameters of a model, in this case, about the data regarding the state of a patient, necessary for CIG execution. Given this initial presentation, the objectives of this book chapter are the following:

- To highlight the advantages of using an already established ontology language to develop a CIG representation model;
- To demonstrate how the procedural and domain knowledge of a CPG can be represented in an ontology and to enumerate the necessary representation primitives to do so;
- Showcase an architecture for a CIG execution environment based on the developed ontology that not only provides the necessary elements for CIG execution, but also a module for the management of incomplete information;
- Propose a method for handling incomplete information regarding data entry points in CIGs in order to produce clinical scenarios of guideline executions.

In order to fulfil the above-mentioned objectives, the chapter is organized as follows. In Sect. 2 we briefly present the current state of the art in CIG research and identify limitations that may pose as research opportunities. We describe a CIG representation model based on Web Ontology Language (OWL) in Sect. 3. In this section we also specify how the model fulfils the basic requirements for the representation of CIGs. Additionally we show and discuss the results of an evaluation performed on the expressiveness of the model. In Sect. 4 we propose a CIG architecture for the execution of guidelines and management of incomplete information. We describe each element and focus mainly on a module based on speculative computation. Finally, in Sect. 5 we draw conclusions from the presented work and make considerations about development perspectives.

2 Modelling and Executing Computer-Interpretable Guidelines

The initial step towards the modelling of clinical recommendations from CPGs in CDSSs was the development of the HELP system [18]. Despite its success, the development of CDSSs based on CIG models only continued in the late 1980s. Arden Syntax [19] was one of the early approaches to CIG modelling. It represents CPGs as sets of independent modules, each one called a *medical logical module* (MLM). A MLM has a structure organized in layers. The first is the *knowledge layer* and contains the clinical knowledge in the form of if-then-else rules whose premises are related to the state of the patient. The *administrative information layer* provides information such as authoring and purpose of the guideline. As for information regarding updates and version, they are kept under the *maintenance layer*. This type of representation handles only a single decision in a CPG and it views guidelines as independent building blocks in the clinical process. However, in reality, the relationships between these knowledge blocks is considerably more complex than this. As such, the most substantial limitation of Arden Syntax is that it is not the most appropriate format for developing complete electronic guideline applications. This model is one of the standards from Health Level 7 (HL7) for medical knowledge.

Another approach to CIG modelling is the Guideline Interchange Format (GLIF3) [20, 21]. It is an approach that organizes CPGs in *steps*, the basic building blocks of the model. The different types of steps are *decision*, *patient state*, *action*, *synchronization* and *branch*. A decision step encodes a decision moment in the CIG workflow, it is used to infer new information based on the state of the patient. The enactment of clinical procedures is encoded in *action* steps, whereas the retrieval of patient information is performed through *patient state* steps. The branching of clinical pathways and their further synchronization is achieved with the *branch* and *synchronization* steps respectively. GLIF3 is an extended formalism that emphasizes the sharing of CIGs across institutions and focuses heavily on the integration of medical knowledge with medical terminologies. However, it relies on subsets of other CIG languages in order to express elements such as temporal constraints on the execution of steps, which may be viewed as a disadvantage.

Temporal constraints are the main focus of the Asbru [22, 23] model for CIGs. Clinical recommendations are represented as decomposable plans with complex temporal patterns and annotations. A plan is a collection of items called *knowledge roles*, which can assume different forms, namely: *preferences*, *plan intentions*, *conditions*, and *effects*. These items define which actions to perform, what is intended with them, what conditions should be gathered to perform them, and what is expected to happen after they are applied. Additionally, Asbru defines a comprehensive set of temporal constructs to represent time. These temporal constructs include constraints on the starting time and ending time of tasks (such as earliest possible start and earliest possible ending), maximal and minimal durations, and cyclical time points (e.g., every morning, every day, etc.). The downside to this is some criticism regarding the complexity of temporal annotations, which seem to be unable to match the knowledge elements in CPGs, particularly those that concern cyclical executions of procedures.

The GuideLine Acquisition, Representation and Execution (GLARE) [24] is a model consisting of atomic and composite *actions*. The *action* is its basic unit, featuring different types such as *work action*, *query action*, *decision* and *conclusion*. The basic building blocks of GLARE follow the structure and meaning of the other models that were already presented. The difference here is the *conclusion* action, which represents the explicit output of a process. GLARE has a comprehensive set of temporal constructs, especially when it comes to cyclical tasks. However, this model is limited in the representation of temporal constraints involving the evolution of the state of a patient.

The Standards Active Guideline Environment (SAGE) [25] is a CIG project that aims to establish a framework for acquiring and sharing guidelines in multiple health care systems. SAGE applies the EON model for the machine-readable formalization of CPGs. Within the model, a CPG consists of *recommendation sets*, represented as a graph of context nodes. Each node is an instance of one of three classes: *action*, *decision* and *routing*. SAGE uses standard terminologies, such as SNOMED-CT [26] and LOINC [27], to provide unequivocal meaning to clinical terms.

All of the above-mentioned modelling approaches have hardly transitioned from the research phase to wide real world implementations, with the most prominent exceptions being Arden Syntax and SAGE. In [28] it is possible to consult an inclusive summary of CIG usage and applications. Moreover, most CIG models require proficiency in programming languages to express logical rules or temporal constraints, which is impractical for health care professionals. With this description of CIG models we intend to show different modelling perspectives, driven by different goals. Whether the focus is placed on decision points, the accurate representation of workflows of procedures, or temporal constraints, there are common features to all models, which we show in the next section. There are other relevant approaches such as PROforma [29] or GUIDE [30]. For an insight on these models and a more detailed overview of CIGs, we urge the reader to consult the works in [8, 9].

In order to properly run CIGs against patient data and obtain clinical recommendations, it is necessary to develop an algorithm, often referred to as execution engine, that analyses the CIG elements given the context provided by the state of the patient. Surrounding the execution engine, there should be a proper system that manages all the workflow from user inputs to the production of recommendations and automatic inferences. Such a system should facilitate the inclusion of CIG advice in different settings [31]. Guideline execution engines such as GLEE [32] for GLIF3, the Spock Engine [33] for Asbru or the GLARE Execution Engine [24] were specifically developed for running CIGs in health care settings. Most of them, including the mentioned examples, exist in the form of client-server applications, with the execution engine placed on the client side. Furthermore, these applications are mostly available as desktop applications, which is an obstacle to their reach and ease of deployment. An extensive review on the computer-based execution of clinical guidelines can be found in [34]. These execution engines could take a better advantage of their knowledge base, i.e., the way in which CIGs are represented and of their knowledge elements, as well as of their patient case base in order to address situations outside of the constraints defined in the guideline workflow. A common example of this is the existence of missing or incomplete information for a data entry point in a guideline, which renders the execution engine unable to produce a clinical recommendation.

3 A CIG Model Based on an Ontology Language

Web Ontology Language (OWL) is an ontology standard developed by the World Wide Web Consortium (W3C) [35]. The second version of OWL (OWL 2) has increased expressiveness and it is built upon formats such as XML, RDF and

RDF-schema. The description logics version of OWL (OWL DL) provides additional vocabulary and its components allow for an easy and expressive representation of the knowledge elements in CIGs. The components that make this possible are:

- Classes: sets that contain individuals described using formal (mathematical) descriptions that state precisely the requirements for membership of the class;
- Individuals: objects of the domain and instances of classes; and
- Properties: binary relations on individuals that may be used to link two individuals (object properties) or an individual to a data element (data properties).

The advantages of this ontology language in the representation of CIGs are related with the internal structure of an ontology in OWL DL. These components are organized in a graph database that is unlike the more common relational and hierarchical databases (nodes and tables). This makes the connection between knowledge components easier and clear. The relationships in OWL assume a greater importance and are the carriers of the semantic content of individuals. Moreover, it is possible to describe or restrain class membership using these relationships and thus accurately delimit their scope.

There are essentially two ways of developing a CIG model. One is consulting domain experts in order to specify the representation primitives. The other is researching different CPGs and determine the information needs of clinical recommendations. The method followed in this work was hybrid in the sense that it included opinions from a health care professional and the observation of guidelines collected from the National Guideline Clearinghouse (NGC).[1] The developed ontology for CPGs was called CompGuide. In it, complex information elements are represented as individuals with multiple object properties connecting them to other individuals, and simple information elements that cannot be further decomposed are represented using data properties. However, simple information that is reusable and will most likely be needed across different points of a CPG is represented as class individuals as well. In this regard the representation is similar to a linked list of procedures. As such, a CPG is represented as an instance of the *ClinicalPracticeGuideline* class. Individuals from this class have a set of data and object properties that enable the representation of descriptive and administrative guideline information such as the name of the guideline, its general description, date of creation, date of last update, version, clinical speciality, category, intended users, and target population.

The CompGuide model was initially presented in [36], where a more detailed description of the model can be found. In the following sections we will present the CompGuide CIG model under the basic requirements for building CIG models defined in Sect. 1.

[1]Available at https://www.guideline.gov/.

3.1 Definition of Workflows of Recommendations

Like other CIG models, CompGuide follows a task network model (TNM) in which all the knowledge elements of CPGs are represented as different tasks. The classes that enable this are:

- *Plan*: a composite task that hosts any number of instances of other tasks, including other plans. An individual of *ClinicalPracticeGuideline* representing a CPG has exactly one global *Plan* within which are hosted all of its tasks. A *Plan* is connected to its first task with the *hasFirstTask* object property;
- *Action*: an activity that should be carried out by a health care professional. It is possible to define subtypes of *Action* via an object property connecting the individual of *Action* to an individual belonging to *Exam*, *MedicationRecommendation*, *NonMedicationRecommendation* and *Procedure*, which specify different types of actions;
- *Question*: a task to obtain information about the health condition of a patient, more specifically about the clinical parameters necessary to follow the guideline. The source of this information can be a human input or an existing database. This class is associated with data properties to define the clinical parameter to be retrieved and the units in which it should be expressed;
- *Decision*: a task representing an inference made in the clinical workflow regarding the state of the patient, based on a set of clinical parameters that act as premises. A common example of this situation would be a diagnosis.

These different classes of tasks, along with the classes used to encode conditions regarding the state of a patient, enclose the domain knowledge of a CPG. The procedural knowledge is defined by the connections that exist between the individuals of these classes. In order to connect individuals belonging to the classes of tasks there is a set of object properties that establish the relative order between them. The definition of sequential tasks is possible with the *nextTask* property. For cases in which, at a splitting point in the clinical workflow, it is necessary to execute one of multiple alternative tasks, the current task is connected to the alternatives with the *hasAlternativeTask* object property. Another situation is when there is a set of tasks that should be executed simultaneously. In this case, the current task is connected to the following tasks with the *hasParallelTask* object property. For both alternative tasks and parallel tasks there are synchronization tasks where the workflow converges to a single execution path.

Although this work draws some inspiration from pre-existing models, such as Arden Syntax, PROforma, GLIF3, Asbru or SAGE, it also introduces different views on the definition of a clinical workflow using the native elements of an ontology language, in this case of OWL.

3.2 Conditions About the State of a Patient and Decision Points

In CompGuide, conditions about the state of a patient are associated with the tasks at which they are verified. In this regard, there are three types of conditions represented by the following classes:

- *TriggerCondition*: this is a condition regarding the clinical parameters of a patient that is used to select an alternative task in the clinical workflow. An alternative task, such as the ones mentioned in Sect. 3.1, has associated trigger conditions, which, when validated, dictate the selection of the task. Trigger conditions can be defined for any type of task;
- *PreCondition*: this condition is used for all types of task to express the requirements of the patient state that must be met before the execution. For instance, when administering some pharmacological agent it should be known that the patient is not allergic to it;
- *Outcome*: this condition puts a restriction to a *Plan* or an *Action* representing the result in terms of the evolution of the state of a patient to be achieved with the task. A typical use for this *Condition* is the specification of therapy goals in an *Action*.

Each of the above-mentioned conditions is connected to an individual of the class *Condition*, which actually allows for the specification of the clinical parameter in question and the value (or range of values) that fulfil the constraint, along with the units in which the parameter is expressed. This class is also important in the definition of the *Decision* class, which consists of a set of individuals of the *Option* class, each one connected to an individual of the *Condition* class. An individual of *Option* specifies a possible conclusion for the *Decision* task, and, in turn it is connected to the *Condition* that supports the option.

Given this exposition about the types of constraints placed on tasks, it is possible to identify four different types of decision points in CompGuide, namely: (i) the selection of an option in a *Decision* task, (ii) the selection of an alternative task based on a *TriggerCondition*, (iii) determining whether a task should be executed or not based on the verification of a *PreCondition*, and (iv) determining if a task was effective based on the verification of an *Outcome*.

3.3 Temporal Constraints on the Execution of Tasks

Time is a crucial dimension in the representation of clinical procedures. This is denoted by the number of CIG representation models that are temporally oriented [22–24]. The temporal constraints in CPGs are used to express a variety of elements that need to be controlled in order to ensure the correct application of recommendations and the proper management of patients. In [37] the temporal aspects of CompGuide are explored and the main classes for temporal representation

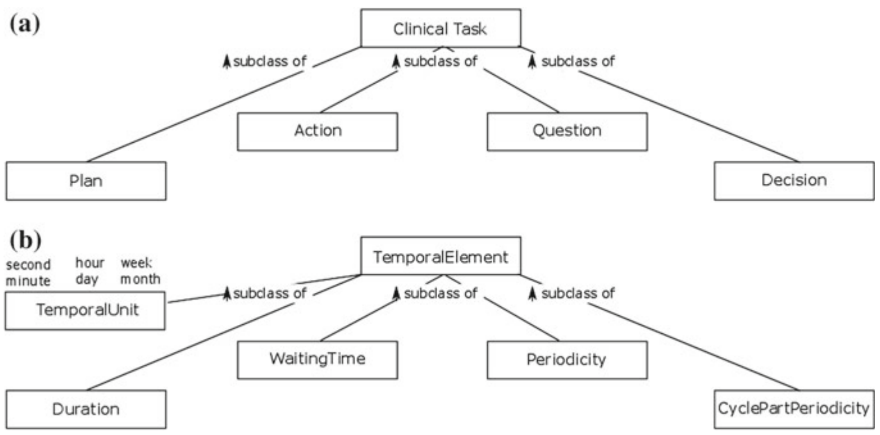


Fig. 1 Main classes for the representation of **a** clinical tasks and **b** temporal patterns

are described. These classes aim to represent the patterns featured in clinical procedures, namely durations, repetitions, periodicities and waiting times. This representation is achieved with subclasses of *TemporalElement* shown in Fig. 1b. The meaning of each one is the following:

- *Duration*: an individual of the class *Duration* allows for the specification of how long an *Action* or a *Plan* should last, since these are the only tasks that may unfold over time. The object property *hasDuration* connects the tasks to the respective *Duration*. This *Duration* can have an exact value or be defined with maximal and minimal values;
- *WaitingTime*: this class stands for a delay in a task used, for instance, to observe an effect on a patient of a previous task. This pattern can be used in any type of task with the *hasWaitingTime* object property;
- *Periodicity*: the class is used to define a cycle of execution for any task. It is possible to define the frequency with which the task is executed and a duration (through the reuse of the *Duration* class) to specify for how long the cycle should last. Alternatively it is also possible to specify the number of repetitions or a stop condition for the task. An individual of *Periodicity* is connected to a task through the *hasPeriodicity* object property;
- *CyclePartPeriodicity*: this class represents a temporal pattern in which there is a nested periodicity, i.e., each cycle of the cyclical task has, itself, a periodicity. Has such, whenever needed, an individual of *CyclePartPeriodicity* is connected to an individual of Periodicity with the *hasCyclePartPeriodicity* object property.

Each temporal pattern, which is the same to say each class, has an associated temporal unit. This is achieved with *TemporalUnit*, an enumerated class that consists of the individuals *second*, *minute*, *hour*, *day*, *week*, *month*, and *year*.

This model offers a comprehensive representation of temporal patterns, at the level of GLARE and more complete than models such as Asbru, GLIF3 and

PROforma [37]. At execution time, the CIG execution engine builds a map of guideline execution for the tasks that have temporal constraints and, then, it performs a series of verifications on the actual starting and ending times of each one.

3.4 Expressiveness of the Representation Model

In order to assess the expressiveness of the ontology elements presented in Sects. 3.1, 3.2, and 3.3 a study was conducted with 14 students from the fourth year of the Integrated Masters in Biomedical Engineering, branch in Medical Informatics, from the University of Minho, in Portugal. The students were familiar with both the CompGuide ontology and the Protégé ontology editor. The study consisted in asking the students to represent a set of CPGs extracted from the NGC. Each student had to represent one CPG using the ontology. After the assignment they were asked to provide feedback in the form of answers to a questionnaire and short comments regarding the expressiveness of the ontology, namely on whether it was possible to completely represent the CPGs using it. The list of CPGs used in the study is showed in Table 1. A much as possible, there was an effort to include CPGs with multiple categories, namely diagnosis, evaluation, treatment, and management. The statements used in the questionnaire complete the general statement: "The CompGuide ontology allowed the representation of:". The answers were provided in a five point Likert rating scale [38] (*1-strongly disagree*, *2-disagree*, *3-neutral*, *4-agree*, *5-strongly agree*). The statements can be seen in Fig. 2, along with the results. The diverging stacked bar presents the total percentage of agreement (calculated as *agree* + *strongly agree*), the total percentage of disagreement (calculated as *disagree* + *strongly disagree*), and the percentage of participants who were neutral (equal to the percentage of the *neutral* category), for each statement.

The statements are related with the basic requirements defined for the representation of CIGs. Items 1–9 can be placed in the definition of workflows of recommendations. Here the level of agreement was at least equal to or above 50%. Indeed, the item about medication prescriptions (item 1) is the one that has the lowest agreement, the highest percentage in the *neutral* category (43%), and the only one that has percentage in the *strongly disagree* category (7%), which is indicative that the corresponding representation primitive for this action may not address all the cases featured in the CPGs. Despite this, both items 2 and 3, which correspond to the representation of other types of actions, seem to correspond to the requirements of guideline representation as they have high percentages of agreement. However, item 4, directly related with the definition of a *Question* is the one that has the highest percentage of disagreement (14%). In the comments provided along with the questionnaire it was mentioned that the *Question* lacked a data property where it would be possible to provide an extended detailed description of the clinical parameters that the task aims to obtain, besides the actual definition of the parameter and units. It is possible to consider that the ontology allows the representation of series of tasks and its internal organization mimics that of the clinical workflows in CPGs. This is

Table 1 List of the guidelines that were used in the study, featuring their name, organization and the number of people assigned to their representation

Clinical practice guideline	Organization	People assigned
Clinical practice guidelines in Oncology - Colon cancer	National comprehensive cancer network	2
Clinical practice guidelines in Oncology - Rectal cancer	National comprehensive cancer network	2
Clinical pratice guidelines in Oncology - Distress	National comprehensive cancer network	2
Clinical practice guidelines in Oncology - Palliative care	National comprehensive cancer network	2
Detection, evaluation, and treatment of high blood cholesterol in adults	national heart lung and blood institute	1
Diagnosing and managing asthma	National heart lung and blood institute	1
Diagnosis, evaluation and management of von willebrand disease	National heart lung and blood Institute	1
Diagnosis and treatment of respiratory illness in children and adults	Institute for clinical systems improvement	1
Diagnosis and management of diabetes	Institute for clinical systems improvement	1
Diagnosis and treatment of ischemic stroke	Institute for clinical systems improvement	1

evident in the high levels of agreement of items 5–9. Overall, the organization of the procedural logic of the guideline and the grouping of tasks in plans was considered to be advantageous, mainly because this helps the delimitation of different diagnoses and treatments. The item that refers to this grouping of tasks, item 6, has an agreement of 100%. Despite this, in the submitted comments concerns were expressed regarding the range of the available subtypes of actions. CPGs also have knowledge encoded as index tables in order to calculate health indexes that are later used in decision making. This type of knowledge could not be represented, which is an aspect to improve. A positive feedback was that, by following the design pattern of the ontology, the participants were able to find redundant elements in the guideline workflows, which did not trigger any kind of event or have any consequences further ahead in the clinical process.

In terms of conditions about the state of a patient, the levels of agreement of items 10, 11, and 12, referring to trigger conditions, pre-conditions, and outcomes was fairly high, which means that these constraints fulfilled, for the most part, their role. The items referring to decision points (items 5, 10, 11, 12) all had high agreement rates, which indicates that they are sufficiently expressive to model decision making in CPGs.

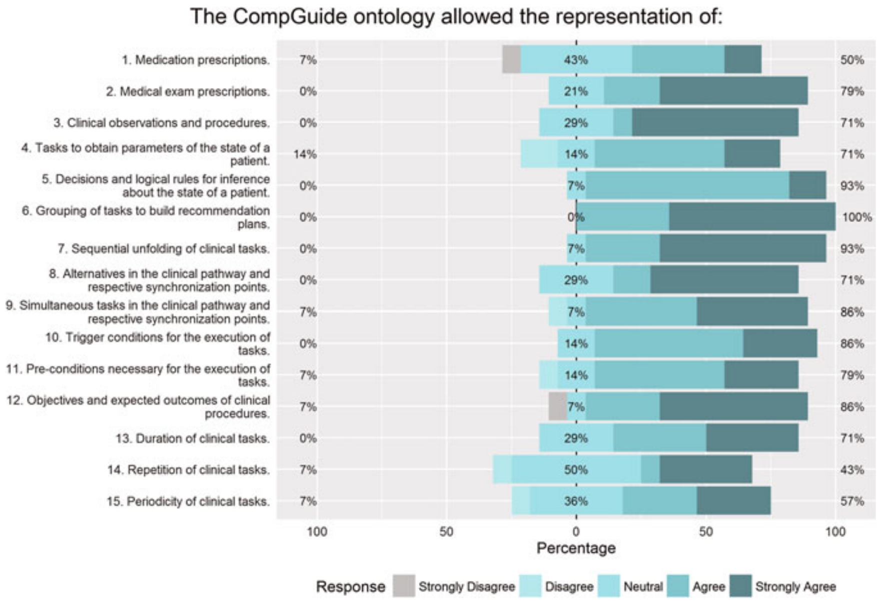


Fig. 2 Diverging stacked bar chart showing the results of the questionnaire to assess the expressiveness of the CompGuide ontology

In the selected guidelines there were no cases of complex temporal patterns. In fact the most common patterns were durations and periodicities. As such, the questionnaire only covers these two temporal elements, along with repetitions, which were also present. The items referring to temporal restrictions, namely items 13–15, have low agreement when compared to the majority of the other items in the chart. Actually, they are among the items that have the highest percentage in the neutral category. It is possible to understand this through the comments of the participants, in which it was pointed out that, although it was possible to represent the temporal patterns in the CPGs, it was necessary to adapt the statements in the guidelines to fit the available constructs.

Given the difficulty and time-consuming nature of the task proposed to the participants, it would be impractical to repeat the study in a larger scale and have access to an entire statistical population of interest. Be as it may, the study provided hints as to what improvements should be made, namely in: the representation of medication prescriptions, the tasks used to retrieve information from the patient, the diversity of actions offered by the ontology, and temporal constraints as a whole. Overall, it is possible to consider that the guidelines used in the survey were accurately represented in the ontology, despite the need for certain adaptations.

4 A CIG Architecture for the Execution of Guidelines and Management of Incomplete Information

As previously stated, a CIG-based CDSS is a knowledge-based system that uses machine-readable versions of CPGs to provide clinical recommendations. As such, the basic elements in the architecture of these systems are a knowledge base containing the CPG recommendations and an execution engine that interprets them. This is a common setting among CIG systems [34]. However, from our point of view, CIG systems should take more advantage of the expressiveness of their CIG models in order to provide additional functionalities that help health care professionals. The architecture that we present herein aims to provide such a functionality, namely one that addresses the problem of incomplete information in decision points such as the ones presented in Sect. 3.2. This problem may arise due to delays in complementary means of diagnosis or the simple impossibility to know the information regarding the clinical parameters. In the following sections we present a description of the elements in the architecture and provide details regarding the knowledge flow throughout the components. Additionally, we present a *speculative module* in charge of managing incomplete information.

4.1 Elements of the Architecture

The proposed architecture can be seen in Fig. 3. Its components are the following:

- *CIG repository*: this component is the knowledge base of the system. It contains *owl* files, each one representing a different CPG.
- *CIG engine*: this component is responsible for interpreting clinical guideline instructions contained in an *owl* file. It identifies different execution situations when analysing the knowledge in the CPG such as the identification of the next task from the control structures in place (sequential tasks, alternative tasks, and parallel tasks), the need to retrieve information from an external source (through *Question* tasks), and the use of that information for automatic inference in decision points;
- *local repository*: a database containing information of other patients retrieved for the data entry points in a CPG.

The problem that the *CIG engine* has to solve is the choice of the next clinical task and infer new information about the state of the patient. The information it uses is obtained from external information sources, which can be health care information systems or simply human agents providing inputs. When choosing the next clinical task from a set of alternatives based on their trigger conditions, when verifying pre-conditions before recommending a task or checking the outcome of an *Action* or *Plan*, such information sources may not be able to provide the necessary information, rendering the *CIG engine* unable to produce a decision. In these cases, a *speculative module*

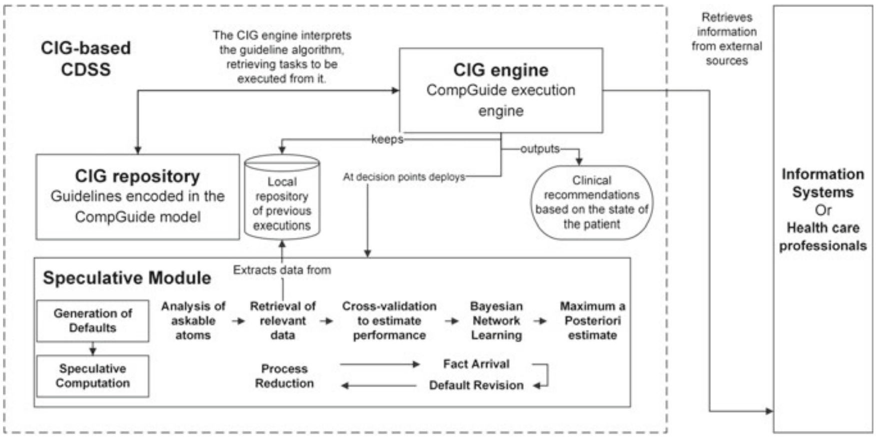

CIG-based CDSS
The CIG engine interprets the guideline algorithm, retrieving tasks to be executed from it.
CIG engine
CompGuide execution engine
Retrieves information from external sources
Information Systems Or Health care professionals
keeps
outputs
CIG repository
Guidelines encoded in the CompGuide model
Local repository of previous executions
At decision points deploys
Clinical recommendations based on the state of the patient
Speculative Module
Extracts data from
Generation of Defaults
Analysis of askable atoms
Retrieval of relevant data
Cross-validation to estimate performance
Bayesian Network Learning
Maximum a Posteriori estimate
Speculative Computation
Process Reduction
Fact Arrival
Default Revision

Fig. 3 Architecture for the execution of CIGs and management of incomplete information

takes action and compensates for these information gaps. It is based on Speculative Computation with Default Revision (SCDR) [37], a logic programming theory that uses default reasoning. As such, this module intervenes when there is a *Decision* task, in the selection of an alternative task based on a *TriggerCondition*, in determining whether a task should be executed or not based on the verification of a *PreCondition*, and in determining if a task was effective based on the verification of an *Outcome*.

4.2 Speculative Module

The *speculative module* has two components. The first is the *generation of defaults* and the second is *speculative computation*. The generation of defaults assumes a supportive role and its function is to produce default values to fill in missing information regarding clinical parameters used in decision points. The *speculative module* then takes these default values and produces clinical scenarios in the form of tentative clinical recommendations and tentative inferences regarding the state of a patient. The speculative module does not ignore the general method for dealing with uncertainty in health care, which is to use past experiences in order to fill in the missing pieces, but it is, instead, a form of using these past experiences in a more flexible way, fitter for CIG-based CDSSs than, for instance, Case-based Reasoning (CBR). The reason for this assumption is that the speculative computation used in the module offers mechanisms to manage information that are not as rigid as the complete CBR cycle. In the work [39] featuring speculative computation, fixed default beliefs are used in speculative computation. However, when applied to a real setting, the default beliefs are highly dependent on the context. The same is to say they depend on the set of circumstances and facts that surround a problem and change over time.

4.2.1 Generation of Defaults

The *generation of defaults* is a set of procedures that seek to acquire the most likely values for the clinical parameters involved in decision points, based on past executions of the same CPG for other patients. In order to take into account possible dependence relationships between the clinical parameters of a decision point, there has to be a default model capable of conveying these relationships in a direct and straightforward way. Bayesian Networks (BNs), for their set of characteristics [40, 41], provide an ideal support for such a task.

The *generation of defaults* is depicted in Fig. 3. It comprises five sequential procedures. The first is the identification of askable atoms, in which the clinical parameters for the decision are identified and isolated. This requires the analysis of the individuals of *Condition* attached to an individual of *Decision*, *Trigger Condition*, *PreCondition*, or *Outcome* in order to extract the parameters whose values are the premises to infer a patient state or a new task in the workflow. Next, the module retrieves relevant data about previous guideline executions regarding the isolated parameters from the local repository. In the following procedure, different BN learning algorithms are used in cross-validation. The objective is to select the one that best conveys the relationships between the parameters. It is possible to do this with a measure of the likelihood of data given the produced model such as the log likelihood loss (*logl*). After the best performing algorithm is selected, a BN is generated. Through a maximum a posteriori estimate (MAP) it is possible to provide the most likely values for the parameters, given the evidence, along with a probability value for the whole distribution [42]. If no evidence is known, i.e., if no value for the set of clinical parameters is available, a MAP can be submitted without evidence, in which case a set of default values is generated for each clinical parameter.

The advantage of using BNs and the MAP estimate to produce defaults is that it is possible to adjust the default values throughout the computation of a clinical recommendation. In fact, that is the principle of SCDR. At the beginning of the decision making process, when no information is available, it is possible to use the BN to generate defaults for all the parameters. However, if suddenly information arrives from the external information sources, it is possible to recalculate the default values by submitting a new MAP query, this time with the value of the known clinical parameter as evidence, thus obtaining default values for the remaining unknown parameters that depend on the piece of information that is actually known at the moment.

4.2.2 Speculative Computation

SCDR acts as the decision framework for the decision points in a CPG. The elements defined in this framework and its operational semantics enable the management of situations of completely unknown information, partially known information and completely known information. The framework is defined as the tuple $\langle \Sigma, \mathcal{E}, \Delta, \mathcal{P} \rangle$, where [37]:

- Σ is a finite set of constants, each one representing an external information source responsible for providing information on clinical parameters;
- $\mathcal{E}$ is a set containing the decision criteria, i.e., the clinical parameters used as premises in the decision points;
- Δ is the default answer set, consisting of default values for the clinical parameters in $\mathcal{E}$, obtained from the generation of defaults;
- $\mathcal{P}$ is a logic program of the form: $H \leftarrow C \parallel B_1, B_2, \ldots, B_n.$, where H is a positive ordinary literal called a head of rule R; C is a set of constraints; and each $B_1, B_2, \ldots, B_n$ is an ordinary literal, or an askable literal. $\mathcal{P}$ results from the mapping of the procedural logic and domain knowledge of the decision point leading to the recommendation of a clinical task or to the inference of a patient state.

SCDR starts with a top goal. The notion of goal is central for it represents what is necessary to achieve in the computation, or, in other words, it is the outcome of the decision. The initial set of beliefs about the state of the patient that the framework uses, if there are no known values, is Δ. Goals and the product of, their reduction, i.e., their matching with rules in $\mathcal{P}$, are kept in processes. They are structures that represent the different alternative computations in the framework. In this regard, there are two types of processes: active and suspended. Active processes are those whose constraints are consistent with the current set of beliefs of the framework. They are regarded as the valid clinical scenarios. Processes that do not fulfil this condition are suspended.

SCDR has a total of three phases: the *process reduction* phase, the *fact arrival* phase, and the *default revision* phase.

Process reduction corresponds to the normal reduction of goals within active processes. It implies the matching of goals with the head of rules in $\mathcal{P}$ and their replacement in the process with the body of the rules they are matched with. If the goal corresponds to one of the clinical parameters, they are reduced with the belief that the framework has about the parameter. If the process is consistent with that, it remains active, otherwise it is suspended. This belief can be either a default or a known value. *Process reduction* continues until an active process with an empty goal is obtained, in which case it is possible to advance a conclusion as an answer to the initial query. If the process is obtained based on default values, it is considered a clinical scenario.

In the *fact arrival* phase there is information about the real value of a clinical parameter that arrives at the *CIG engine* and is passed on to the *speculative module*. This information is handled in the following way. The beliefs of the framework are updated and the default value for the clinical parameter is replaced with the real value. It can be the case that the real value is the same as the default. As a result, all the processes, both active and suspended, are revised. Those that are consistent with the real value stay or become active and those that are inconsistent are discarded because the new information is considered to bee definitive and, thus, cannot possibly be revised again. Following this phase there is always a round of *process reduction*.

The *default revision* phase results from the verification of changes that the newly arrived information may cause in the default values of the other clinical parameters. As such, a new MAP query is submitted having the known information as evidence. If the retrieved values for the remaining clinical parameters are different from the previous defaults, then they replace the old defaults in the beliefs of the framework. Both the active and the suspended processes are revised and only the processes that are consistent with the new defaults remain or become active, the rest is suspended. This *default revision* phase ensures that the active processes move progressively to the actual true recommendation, since the framework is able to respond to the arrival of information by adjusting the other defaults to values that are closer to the respective real values, according to the probability distribution of the underlying BN. After default revision, another round of process reduction occurs.

4.3 Generation of Clinical Scenarios

Although in the presented architecture there is a clear separation between the knowledge base of the system and its execution engine, the design of the CompGuide ontology determines to a great extent the procedures of the execution engine. The ontology defines a set of decision points that are identified by the *CIG engine* and then mapped to the SCDR framework, which structures the reasoning process and endows the system with dynamic belief revision capabilities. This is seen when the active processes, standing for clinical scenarios, resulting from the different phases of SCDR are changed and updated into new active processes and, thus, new scenarios. In situations where the results of clinical exams may take some time to be known or may turn out to be inconclusive the effect of speculative computation yields tentative answers that enhance the capacity of health care professionals to make decisions.

If one considers the complexity of a guideline such as the Clinical Practice Guidelines in Oncology Colon Cancer [43] (which was one of the CPGs used in the study), with multiple data entry points in the form of individuals of *Question*, followed by decision points consisting in the choice of alternative tasks (such as the clinical workflow showed in Fig. 4), it is possible to use the *speculative module* on each one and, through it, present the most likely execution threads by summing the computation of these choices. Figure 4 shows an example of how this could be applied to a clinical workflow represented in CompGuide. For every decision point in the algorithm the *speculative module* runs on top of the procedural knowledge provided by the ontology. Assuming that information is missing in each *Question* task, the *speculative module* formulates a probable choice for the next task at Question1, Question2, and Question3. Then, by grouping the proposals, it is possible to build a tentative execution path, which is shown by the dashed lines in the figure. This would be useful for a practitioner as it would provide him a map of the potential evolution of a patient, thus giving him time to devise countermeasures if it shows that the treatment is following an undesirable direction.

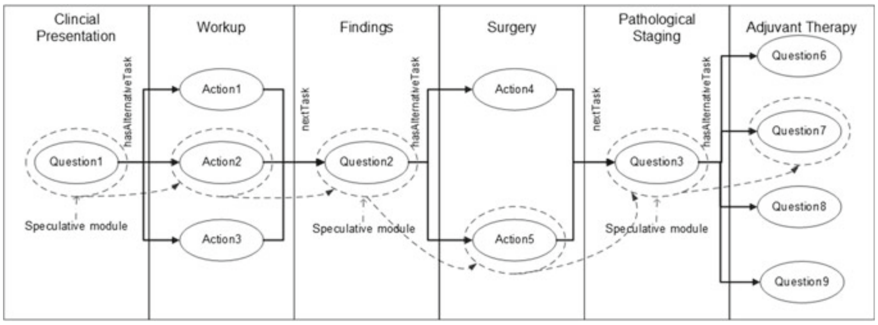


Fig. 4 Stages of the NCCN Guideline for Colon Cancer showing a symbolic representation of tasks as questions and actions throughout the clinical process: clinical presentation, workup, findings, surgery, pathological staging and adjuvant therapy. The *dashed trace* shows an execution path obtained from the *speculative module*

5 Conclusions and Development Perspectives

It was established that the CompGuide representation model was able to provide enough expressiveness for a set of CPGs that included multiple categories and specialties, in terms of the basic requirements for a CIG model. The CompGuide ontology enables the creation of modular knowledge components and thus their reuse in different points of a CPG. By analysing and identifying the decision points in a CPG represented according to CompGuide, it is possible to map the procedural and domain knowledge of these points into a speculative computation framework that manages incomplete information. This is the basis of a *speculative module*, responsible for building clinical scenarios, based on default values retrieved from BNs. The use of probabilities, and more specifically BNs, is motivated by the notion that the knowledge we have about the world is imperfect and that, through a Bayesian approach, it is possible to get a degree of belief that something may be the case. The inclusion of the *speculative module* is a differentiating factor from other CIG execution engines such as GLARE, GLEE and SAGE [34], which only execute their encoded rules, without additional functionalities.

There are, however, aspects to improve in the ontology in terms of knowledge representation, namely in the definition of action tasks and temporal constraints. Additionally, the utility of the clinical scenarios is not considered in the SCDR framework, but it is an important dimension since it useful for the health care professionals to know how reliable a scenario is. As such, the extension of the framework to accommodate the computation of utilities based on the probabilities provided by the BN model is a development perspective that we will follow.

Acknowledgements This work has been supported by COMPETE: POCI-01-0145-FEDER-007043 and FCT Fundação para a Ciência e Tecnologia within the Project Scope UID/CEC/00319/2013. The work of Tiago Oliveira is supported by a FCT grant with the reference SFRH/BD/85291/

2012. This work was partially developed during an internship program of the National Institute of Informatics (NII) of Japan by Tiago Oliveira.

References

1. Engelmore, R.S.: Artificial intelligence and knowledge based systems: origins, methods and opportunities for NDE. In: Review of Progress in Quantitative Nondestructive Evaluation, vol. 6 A, pp. 1–20 (1987)
2. Studer, R., Benjamins, V.R., Fensel, D.: Knowledge engineering: principles and methods. Data Knowl. Eng. **25**(1–2), 161–197 (1998)
3. Kalogeropoulos, D.A., Carson, E.R., Collinson, P.O.: Towards knowledge-based systems in clinical practice: development of an integrated clinical information and knowledge management support system. Comput. Methods Programs Biomed. **72**(1), 65–80 (2003)
4. Miller, M., Kearney, N.: Guidelines for clinical practice: development, dissemination and implementation. Int. J. Nurs. Stud. **41**(7), 813–821 (2004)
5. Silberstein, S.: Clinical practice guidelines. J. Neurosurg. Pediatr. **25**(10), 765–766 (2005)
6. Woolf, S.H., Grol, R., Hutchinson, A., Eccles, M., Grimshaw, J.: Potential benefits, limitations, and harms of clinical guidelines. BMJ Br. Med. J. **318**(7182), 527–530 (1999)
7. Toker, A., Shvarts, S., Perry, Z.H., Doron, Y., Reuveni, H.: Clinical guidelines, defensive medicine, and the physician between the two. Am. J. Otolaryngol. Head Neck Med. Surg. **25**(4), 245–250 (2004)
8. Peleg, M.: Computer-interpretable clinical guidelines: a methodological review. J. Biomed. Inform. **46**(4), 744–763 (2013)
9. Oliveira, T., Novais, P., Neves, J.: Development and implementation of clinical guidelines: an artificial intelligence perspective. Artif. Intell. Rev. 999–1027 (2014)
10. Latoszek-Berendsen, A., Tange, H., van den Herik, H.J., Hasman, A.: From clinical practice guidelines to computer-interpretable guidelines. A literature overview. Methods Inf. Med. **49**(6), 550–570 (2010)
11. Codish, S., Shiffman, R.N.: A model of ambiguity and vagueness in clinical practice guideline recommendations. In: AMIA Annual Symposium Proceedings, vol. 2005, p. 146 (2005)
12. de Clercq, P.A., Blom, J.A., Korsten, H.H.M., Hasman, A.: Approaches for creating computer-interpretable guidelines that facilitate decision support. Artif. Intell. Med. **31**(1), 1–27 (2004)
13. Novais, P., Oliveira, T., Neves, J.: Moving towards a new paradigm of creation, dissemination, and application of computer-interpretable medical knowledge. Prog. Artif. Intell. 1–7 (2016)
14. Lipshitz, R., Strauss, O.: Coping with uncertainty: a naturalistic decision-making analysis. Organ. Behav. Hum. Decis. Process. **69**(2), 149–163 (1997)
15. Babrow, A., Kasch, C., Ford, L.: The many meanings of uncertainty in illness: toward a systematic accounting. Health Commun. **10**(1), 1–23 (1998)
16. Mishel, M.H.: The measurement of uncertainty in illness. Nurs. Res. **30**(5), 258–263 (1981)
17. Han, P.K.J., Klein, W.M.P., Arora, N.K.: Varieties of uncertainty in health care: a conceptual taxonomy. Med. Decis. Mak. Int. J. Soc. Med. Decis. Mak. **31**(6), 828–38 (2011)
18. Gardner, R.M., Pryor, T., Warner, H.R.: The HELP hospital information system: update 1998. Int. J. Med. Inform. **54**(3), 169–182 (1999)
19. Samwald, M., Fehre, K., de Bruin, J., Adlassnig, K.P.: The Arden Syntax standard for clinical decision support: experiences and directions. J. Biomed. Inform. (2012)
20. Peleg, M., Boxwala, A.A., Bernstam, E., Tu, S., Greenes, R.A., Shortliffe, E.H.: Sharable representation of clinical guidelines in GLIF: relationship to the Arden Syntax. J. Biomed. Inform. **34**(3), 170–181 (2001)
21. Boxwala, A.A., Peleg, M., Tu, S., Ogunyemi, O., Zeng, Q.T., Wang, D., Patel, V.L., Greenes, R.A., Shortliffe, E.H.: GLIF3: a representation format for sharable computer-interpretable clinical practice guidelines. J. Biomed. Inform. **37**(3), 147–161 (2004)

22. Shahar, Y., Miksch, S., Johnson, P.: The Asgaard project: a task-specific framework for the application and critiquing of time-oriented clinical guidelines. Artif. Intell. Med. **14**(1–2), 29–51 (1998)
23. Seyfang, A., Miksch, S., Marcos, M.: Combining diagnosis and treatment using ASBRU. Int. J. Med. Inform. **68**(1–3), 49–57 (2002)
24. Terenziani, P., Montani, S., Bottrighi, A., Torchio, M., Molino, G., Correndo, G.: The GLARE approach to clinical guidelines: main features. Stud. Health Technol. Inform. **101**(3), 162–166 (2004)
25. Tu, S.W., Campbell, J.R., Glasgow, J., Nyman, M.A., McClure, R., McClay, J., Parker, C., Hrabak, K.M., Berg, D., Weida, T., Mansfield, J.G., Musen, M.A., Abarbanel, R.M.: The SAGE guideline model: achievements and overview. J. Am. Med. Inform. Assoc. **14**(5), 589–598 (2007)
26. Cornet, R., Schulz, S.: Relationship groups in SNOMED CT. Stud. Health Technol. Inform. **150**, 223–227 (2009)
27. Dugas, M., Thun, S., Frankewitsch, T., Heitmann, K.U.: LOINC(R) codes for hospital information systems documents: a case study. J. Am. Med. Inform. Assoc. **16**(3), 400–403 (2009)
28. Open Clinical: Methods and tools for representing computerised clinical guidelines. http://www.openclinical.org/gmmsummaries.html (2013)
29. Fox, J., Ma, R.T.: Decision support for health care: the PROforma evidence base. Inform. Prim. Care **14**(1), 49–54 (2006)
30. Ciccarese, P., Kumar, A., Quaglini, S.: NEW-GUIDE: a new approach to representing clinical practice guidelines. In: Advances in Clinical Knowledge Management (Figure 1), pp. 15–18 (2002)
31. Costa, R., Neves, J., Novais, P., Machado, J., Lima, L., Alberto, C.: Intelligent Mixed Reality for the Creation of Ambient Assisted Living, pp. 323–331. Springer, Berlin (2007)
32. Wang, D., Peleg, M., Tu, S.W., Boxwala, A.A., Ogunyemi, O., Zeng, Q., Greenes, R.A., Patel, V.L., Shortliffe, E.H.: Design and implementation of the GLIF3 guideline execution engine. J. Biomed. Inform. **37**(5), 305–318 (2004)
33. Young, O., Shahar, Y.: The spock system: developing a runtime application engine for hybrid-asbru guidelines. Artif. Intell. Rev. **3581**(1), 166–170 (2005)
34. Isern, D., Moreno, A.: Computer-based execution of clinical guidelines: a review. Int. J. Med. Inform. **77**(12), 787–808 (2008)
35. McGuinness, D.L., Van Harmelen, F.: OWL Web Ontology Language Overview. https://www.w3.org/TR/owl-features/ (2004)
36. Oliveira, T., Novais, P., Neves, J.: Representation of clinical practice guideline components in OWL. In: Pérez, J.B., Hermoso, R., Moreno, M.N., Rodríguez, J.M.C., Hirsch, B., Mathieu, P., Campbell, A., Suarez-Figueroa, M.C., Ortega, A., Adam, E., Navarro, E. (eds.) Advances in Intelligent Systems and Computing, vol. 221, pp. 77–85. Springer International Publishing, Berlin (2013)
37. Oliveira, T., Satoh, K., Novais, P., Neves, J., Hosobe, H.: A dynamic default revision mechanism for speculative computation. Auton. Agents Multi-Agent Syst. 1–40 (2016)
38. Jamieson, S.: Likert scales: how to (ab)use them. Med. Edu. **38**(12), 1217–1218 (2004)
39. Hosobe, H., Satoh, K., Codognet, P.: Agent-based speculative constraint processing. IEICE Trans. Inf. Syst. **E90–D**(9), 1354–1362 (2007)
40. Visscher, S., Lucas, P.J.F., Schurink, C.A.M., Bonten, M.J.M.: Modelling treatment effects in a clinical Bayesian network using Boolean threshold functions. Artif. Intell. Med. **46**(3), 251–266 (2009)
41. Van der Heijden, M., Lucas, P.J.F.: Describing disease processes using a probabilistic logic of qualitative time. Artif. Intell. Med. **59**(3), 143–155 (2013)
42. Korb, K., Nicholson, A.: Bayesian Artificial Intelligence, 2nd edn. CRC Press, London (2003)
43. Benson, A., Bekaii-Saab, T., Chan, E., Chen, Y.J., Choti, M., Cooper, H., Engstrom, P.: NCCN Clinical Practice Guideline in Oncology Rectal Cancer. Techical report, National Comprehensive Cancer Network. http://www.nccn.org/professionals/physician_gls/f_guidelines.asp (2013)

Metamarket – Modelling User Actions in the Digital World

Adrian Giurca

Abstract We present Metamarket (`http://metamarket.info`), an ontology of user actions on the Web as a foundation for understanding user preferences from Web activities and its relations with the state of the art. Metamarket is implemented using the Web Ontology Language (OWL) and is the base of a platform offering Linked Data access for the purpose of market research allowing intelligent applications to enhance local business sales and make new business insights possible. Particularly, the use of user generated content, i.e., digital works in which data from one or more sources is combined and presented in innovative ways, is a great way to expose this value. Although there are many approaches to publishing and using consumer data, we believe Linked Data is a key solution of many of the challenges and can lower the cost and complexity of developing these applications. In addition, Metamarket can be used to develop intelligent UX applications, responsive to user needs, and it can be extended towards modeling emotions.

1 Introduction

eCommerce helps businesses to contact a large number of potential customers but nowadays, customers are active on many communication channels. Enabling flexible eCommerce requires complex processes that combine web design and development, online publishing, search engine optimization, behavioral advertising, reputation management, social media management and other avenues to create a long-term positive presence for a person, organization, or product in the digital world.

This work was supported by the Chair of Soil Protection and Recultivation of Brandenburg University of Technology Cottbus-Senftenberg and Binarypark UG.

A. Giurca (✉)
Brandenburg University of Technology Cottbus-Senftenberg,
Konrad-Wachsmann-Allee 6, 03046 Cottbus, Germany
e-mail: giurca@b-tu.de

G.J. Nalepa and J. Baumeister (eds.), *Synergies Between Knowledge Engineering and Software Engineering*, Advances in Intelligent Systems and Computing 626, https://doi.org/10.1007/978-3-319-64161-4_11

Understanding consumer preferences is typically done by economics quantitative research solutions typically based on consumer surveys. However, the Social Web is dominated by rating systems such as the ones of Facebook ("Like", "Love", "Ha ha", "Wow", "Sad", "Angry"), YouTube ("Like", "Dislike") or the Amazon product review 5-star rating [1]. All these systems try to pool the preferences of different agents so as to best reflect the wishes of an agent and the population as a whole. The main assumption derives from the theory of social choice i.e., agents have preferences, and do not try to camouflage them in order to manipulate the outcome to their personal advantage.

In the main stream, ontologies were developed to facilitate knowledge sharing and reuse. They are so popular due to what they promise: a common understanding of a domain that can be communicated between application systems. They are formal specification of conceptualizations that provide a shared and common understanding of a domain, an understanding that can be communicated across application systems.

We introduce Metamarket an usable activity ontology to create a conceptual for inferring user preferences related to the online content. After investigating a number of available activity ontologies the main finding was that the word "activity" is multi-semantic therefore many of the developed ontological resources were not coping with our needs. Metamarket aims to stay interoperable with the Schema.org[1] vocabulary (Fig. 1).

Our goal is to provide a simple and extensible conceptual framework and tools for understanding user preferences on the web. In a nutshell, to understand user preferences it is necessary to

1. Design and implement activity models in a way that allows software systems/agents to conduct reasoning and manipulation. This is Metamarket.
2. Provide tools to monitor and capture the consumer behavior along with the state of the environment.
3. Provide tools to process the perceived information towards generating a high-level abstraction of context or situation.
4. Use tools that carry out preference recognition to perform recommendations.

Unfortunately, we were not able to adapt any of the investigated ontologies such as such as DnS [2], NeOn [3], context modeling [4], AMOn [5], WAP [6], Date [7], Activities of Daily Living (ADL) [8], ODP [9], ICA [10], LODE [11], OWL-ACT [12], SNAP [13] to our goals. The difficulty of reusing was either because they were already too complex for our goal or they were defining classes and properties not necessary in our use case. Therefore we decided to model Metamarket.

[1]Schema.org (sponsored by Google, Microsoft, Yahoo and Yandex) is a collaborative, community activity with a mission to create, maintain, and promote schemas for structured data on the Internet. The vocabulary, covering entities, relationships between entities and actions, is developed by an open community process and can be used with many different encodings, including RDFa, Microdata and JSON-LD. Many applications from Google, Microsoft, Pinterest, Yandex and others already use these vocabularies to power rich, extensible experiences. Binarypark provides `http://getschema.org` a mediawiki website with a lot of examples and best practices on how to use this vocabulary.

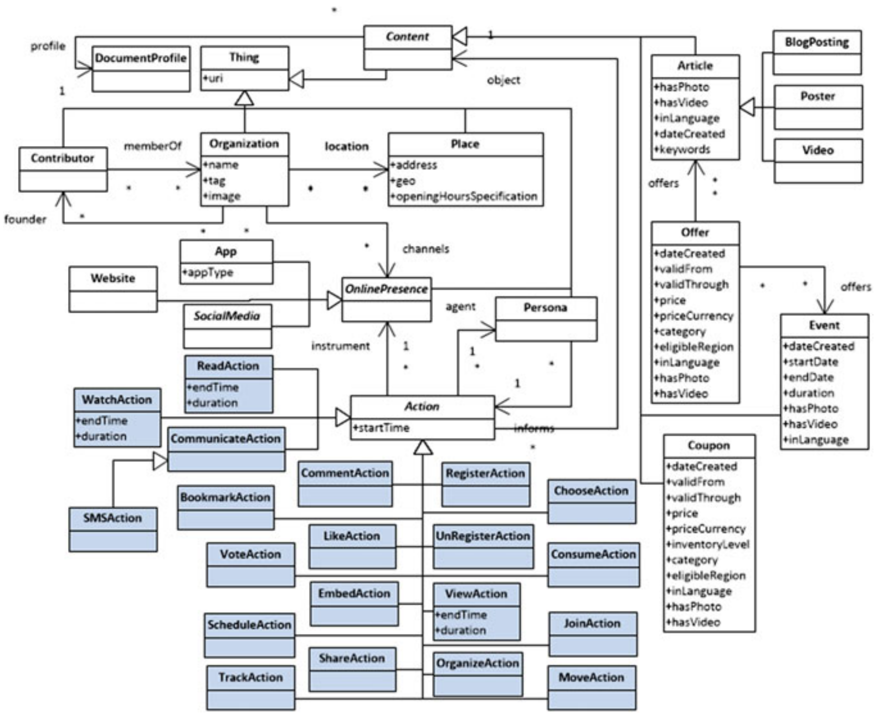


Fig. 1 UML diagram of the metamarket ontology

The rest of the chapter is organized as follows:

Section 2, *A Brief Overview of Activity Ontologies* describes existing work on Activity Ontology and their relationship with our case. We provide information about the core ontology classes and properties, their intended usage and meaning as well as relationships between them.

Section 3, the core of the chapter, presents *Metamarket* an ontology of atomic actions performed by actors on digital communication channels. Metamarket aims to provide a conceptual foundation towards understanding consumer preferences from his digital activities. In a nutshell, Metamarket defines 46 classes and 28 properties. 20 classes define agent actions, 7 classes define entities published on online presence channels, but an application layer may extend all these according with supplementary requirements.

Conclusions are provided in Sect. 4, *Final Remarks*.

2 An Overview of Events and Activity Ontologies

Context-Aware Health Promotion Application [4], is a top level activity ontology providing classes aiming at recognition of user activities in the context of usage of health applications. This ontology started to conceptualize a framework of context sensors such as phone-embedded sensors, body sensors, and other environmental sensors, towards providing the foundation for inferring persuasive user recommendations with the main goal to increase motivation in doing health related activities. While their use case is very closed to our goals, we found very difficult to use their findings because of two main issues: (1) the to high level of abstraction of their concepts and (2) unavailability of this ontology in a standard Semantic Web format such as OWL and/or RDFS (Fig. 2).

Dementia Ambient Care (DnS) [2], is an ontology designed to detect activities of daily living (ADL) for improving the healthcare support for elderly population, particularly to detect potentially dangerous behaviours. Apart of the conceptual modeling they employed a physical sensor system such as contact sensors, cameras, microphones to collect data from different ambient sources.

The Activity Model ONtology (AMOn) [5], provides OWL representation of the concepts from the activity model. It is a multi-layered and modular ontology aiming for modeling of many real world activities. The ontology includes essential aspects of interpersonal communication activities, focusing on a specific activity (such as job interview, body language). AMOn provides the following modules

- `AbstractActivityModel`,
- `BootstrappedBuddyMentoringActivityModel`,
- `BootstrappedJobInterviewActivityModel`,
- `DyadicInterpersonalCommunicationActivityModel`,
- `InterpersonalCommunicationActivityModel`.

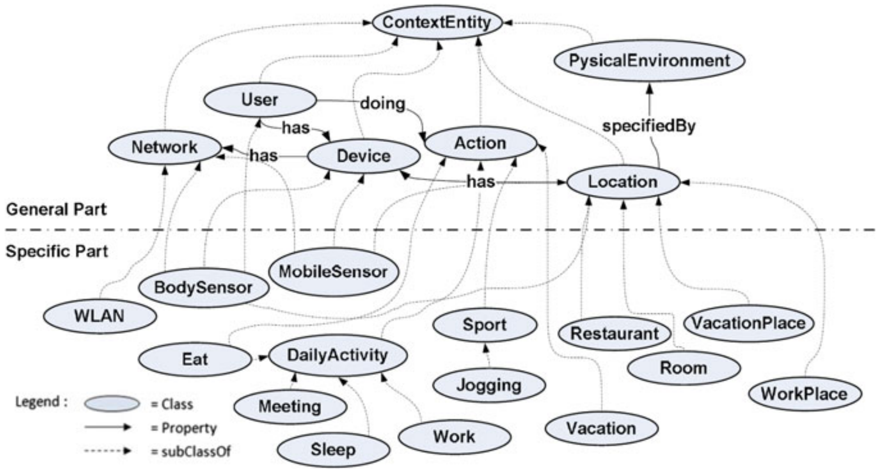


Fig. 2 Context health classes [4]

The follow up Amon+ enrich the initial version to conceptualize cultural variations in interpersonal communication. This version has 70 classes, 16 object properties. However, AMOn is not suitable for understanding activities on the Web as the defined concepts are more related to psychological intercultural communication acts.

The NeOn toolkit [3], is a state-of-the-art ontology engineering environment providing support for the ontology engineering life-cycle. The ontology covers a variety of ontology engineering activities, such as `Annotation`, `Documentation`, `Development`, Human-Ontology Interaction, `Knowledge Acquisition`, `Management`, `Modularization`, `Matching`, `Reasoning`, and `Reuse`. NeOn defines a comprehensive set of activities (over 50 activities) to be used in engineering tasks, including ontology engineering their main goal, but we cannot use them to achieve user activity recognition.

WAP [6] is devoted to discovery of workflow activity patterns. The authors goal was to provide a discovery framework to be used in a wide range of business processes towards detecting specific patterns (called activities). The main reason of their effort was automation of this process because the expert-based, human discovery of workflow activity patterns in business processes is not a trivial task, mostly because the purpose and use of any given activity can be subject of different interpretations. However, the work was not suitable for our purpose as the ontology defines four main concepts `Activity`, `Message`, `Signal`, and `Pattern`.

Activity Ontologies for Intelligent Calendar Applications (ICA) [10] is a significant work as calendar applications are frequently used by millions of people world-wide. ICA is built on traditional electronic calendars, by empowering them with efficient scheduling techniques. Obviuosly they needed a much more detailed description of events beyond existing XML based formats to describe events, such as iCalendar, which do not provide enough context to describe a rich event. ICA has been designed to describe events, their temporal aspects, as well as the users constraints and preferences about these events. ICA defines (Fig. 3):

- 114 OWL classes, including Meeting, Calendar, TimeProgramOrganizer, DateInterval, Interval, ComboInterval,
- 127 subclass relations,
- 16 equivalent classes,
- 12 object properties,
- 3 data properties, and
- 207 named individual objects.

and use resources from other ontologies such as FOAF [14] (used to represent user related information, like names, addresses, contact info), OWL-Time ontology [15] (to define time related information, such as duration, dates, intervals), Geo OWL ontology [16] (to describe spatial information, such as the latitude or the longitude of a specific location).

Linking Events with Media (LODE) [11, 17] defines a minimal model of most useful properties for describing events. The goal of this ontology is to enable interoperability from the perspective of *what* happened (`lode:Event`), *where* did it

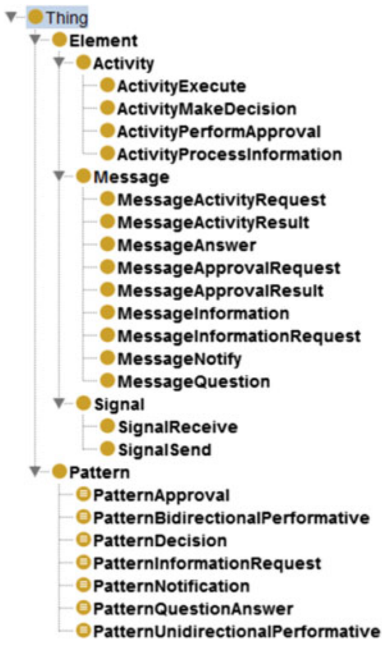


Fig. 3 Class hierarchy for the WAP ontology [6]

happen (`lode:atPlace`, `lode:inSpace`), *when* did it happen (`lode:atTime`, `lode:circa`), and *who* was involved (`lode:involvedAgent`).[2]

It is worth to mention a very similar model proposed by Van Hage et al. [18], a simple RDF model for describing any event based on its type (what), its location (where), its participants (who) and its temporal information (when), allowing for a large category of questions to be answered using simple SPARQL queries (Fig. 4).

Date [7] is a formal ontology to define time dates on the Web as the representation of dates and their relationship to time and duration has long been recognized as an important problem in commonsense reasoning. Although this formalism introduced a complete approach of duration, time and date it is far limited for our goal.

Activities of Daily Living (ADL) [8] aims to provide an overview addressing the state-of-the-art in the area of activity recognition, in particular, in the area of object-based activity recognition. However, their work is focusing on real life activity models such as `MakeHotDrink` or `MakeMeal` as such there is little chance to be usable for consumer activity recognition on the web (Fig. 5).

DESO [19] and the follow up ABDES [20] are significant work on complex events modeling and agent-based simulation. DESO is a top level ontology for discrete event system modeling derived from the foundational ontology UFO. The main purpose of DESO is to provide a basis for evaluating discrete event simulation languages. DESO is divided into the run-time ontology DESO-I defining the necessary events an event simulator may deal with, and design-time ontology DESO-U describing the

[2]See `http://linkedevents.org/ontology/` for the latest specification.

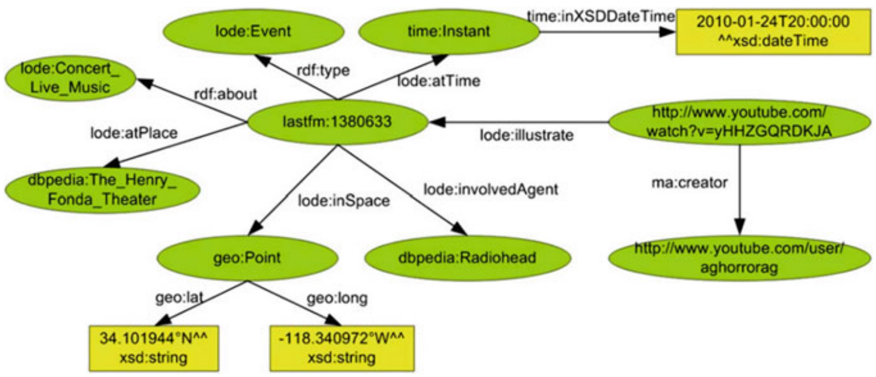


Fig. 4 The radiohead Haiti relief concert described with LODE [11]

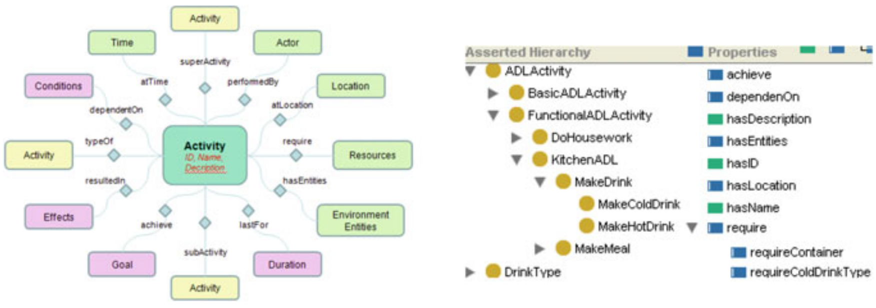


Fig. 5 Activities of daily living [8]

entity types. This distinction comes from the MDA approach and it is very useful from the perspective of software engineering, offering the developer the complete framework to design and implement a discrete event simulator. A very important feature is the ontologies are modeled with UML as such making them easy familiar with the developers (Fig. 6).

Both ontologies are top level or foundational while our goal was to consider an immediate usable one along with a tight requirement to stay aligned with concepts of Schema.org. However, their approach is very relevant for software engineers and it is worth to research on Metamarket aligned with DESO.

ODP [9] has been formally encoded[3] using the Web Ontology Language (OWL) which is a great achievement as makes it available for reasoning on the Web. The core relationships are temporal and somehow cope with our development of Metamarket. Spatiotemporal relations are captured through the properties `takesPlaceAt` (indicates the place where an activity happens), `hasStart`, `hasEnd` (the time an activity starts/ends) as well as `hasDuration` (the time period that an activ-

[3]See http://descartes-core.org/ontologies/activities/1.0/ActivityPattern.owl.

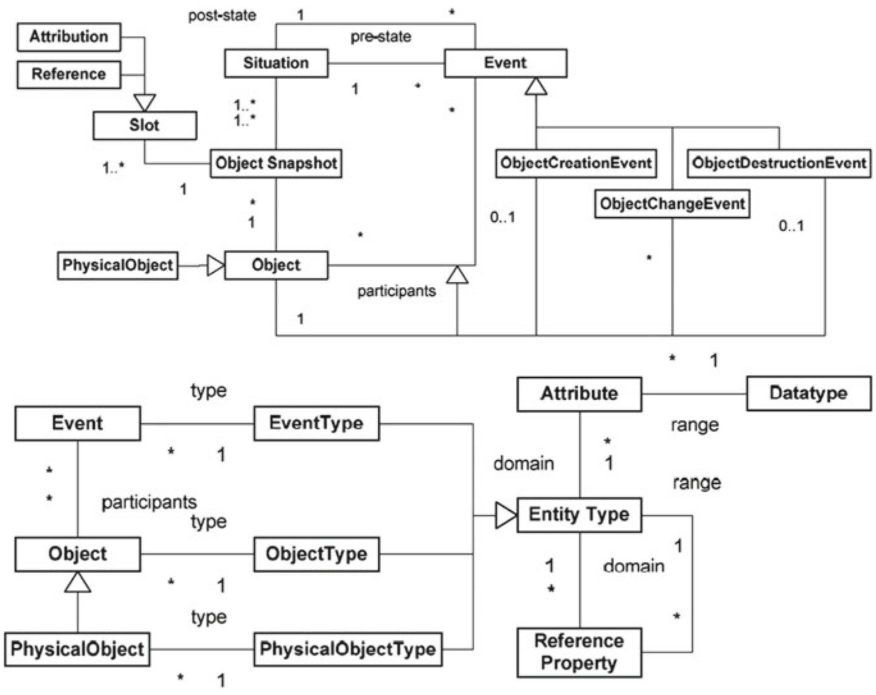


Fig. 6 Individuals and universals in discrete event simulation [19]

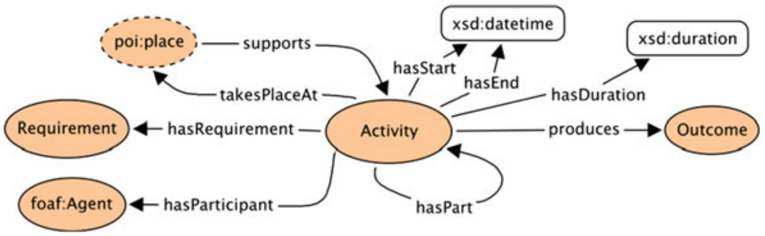


Fig. 7 A schematic view of the activity ODP [9]

ity lasts. Obviously duration equals the difference between the start and end time) (Figs. 7, 8, 9).

Although Metamarket is more comprehensive and equally more narrow on Web activities we found Activity ODP a great source of inspiration. In addition we would like to point out `http://ontologydesignpatterns.org` where the reader can find a great source of professionally designed Ontology Patterns.

OWL-ACT [12] defines an Activity ontology similar with ICA [10] There is a top level Activity class and a taxonomy of different sub-activities based on specific constraints applicable to that kind of activity, e.g., the number of participants (`SocialActivity`), or distinct properties of such subtypes (e.g. `CarnivalParty`). However, their total work focuses on identifying indoor com-

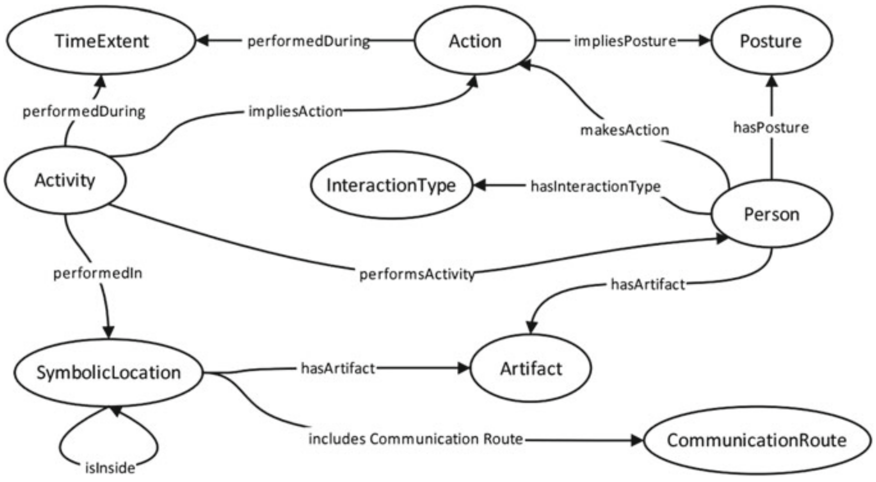


Fig. 8 OWL 2 activity ontology: core classes and properties [12]

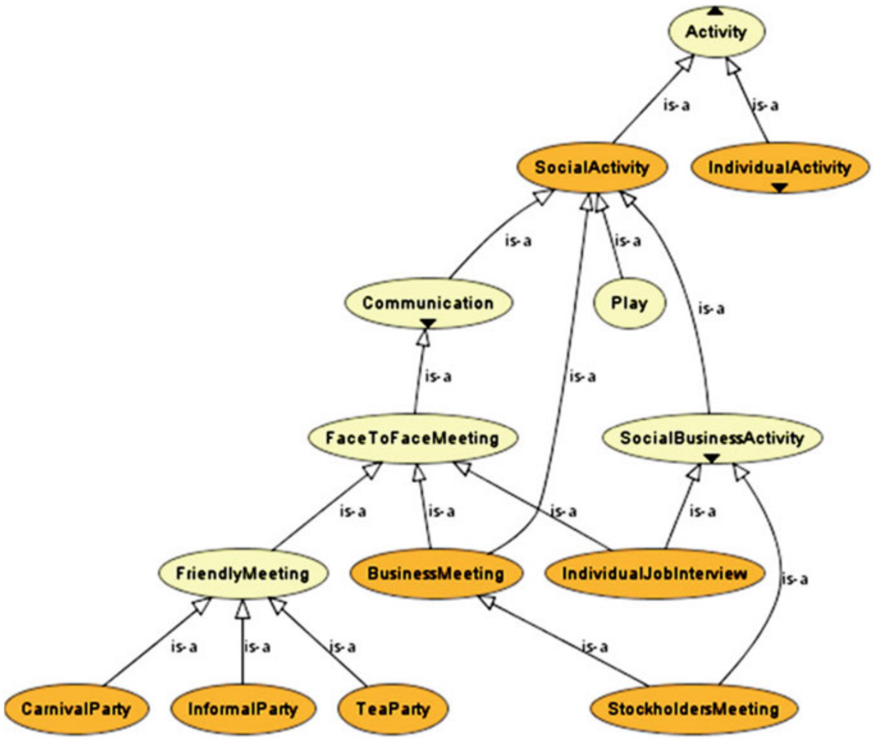


Fig. 9 Social activities [12]

plex human activities, from data provided by smart home sensors it is quite difficult to apply in the Web context.

SNAP [13] is an e-commerce ontology developed for automated recommendation systems for the domains of banking, insurance, and telephony. SNAP stands for Situations, Needs, Actions, and Products.

TO DO Obviously there is much more research on modeling Web objects and their interactions than we provided here.

3 Metamarket

Metamarket[4] is an ontology of activities on digital communication channels providing a conceptual foundation towards understanding consumer preference from his digital activities.

Schema.org does not offer up to date OWL description of its resources and is beyond the scope of this work to discuss this issue. However, using http://schema.org/docs/schemaorg.owl the reader will get a historical OWL version. Despite that the OWL description it is quite intuitive following Schema.org website.

3.1 Metamarket Entities

Metamarket features only a limited number of entities, particularly those interesting for our case but it can be easily extended with many more according with other applications needs.

Metamarket uses the following prefixes:

```
@PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
@PREFIX s: <http://schema.org/>
@PREFIX b: <http://metamarket.info/>
@PREFIX xs: <http://www.w3.org/2001/XMLSchema/>
```

Metamarket defines the following entities:

- `Organization` – describes an organization from the perspective of its location members and digital channels it manages.
- `Place` – a class used for describing a location.
- `Persona` – a class to represent a visitor of an online presence channel.
- `Contributor` – a person which is content creator in the ecosystem
- `OnlinePresence` (with subclasses `Website`, `SocialMedia` and `App`) – defines the digital channels. An organization can publish content on one or more

[4] See http://metamarket.info.

channels, can monitor all online presence channels and can reply on people's reactions on a particular channel.
- `Content` (with subclasses `Article`, `Event`, `Offer`, `Coupon`) – classes describing various kinds of content that can pe published on the online presence management channels.

All Metamarket content is subclass of the abstract class `Content`. This class defines the domain of property `profile`. The value of this property is of type `DocumentProfile`. A document profile is a derived class that can be computed from the document content. Usually is subject of a summarization and classification algorithm. Metamarket let the users of the ontology to define this class according with their own requirements. Usually, Consumer/Document profile is characterized by a set of properties, each property with a range of values. Metamarket does not impose any definition of such profile. Example of properties are (see also [1]):

- Categorical Properties:
  - *nominal properties*, such as `gender` with a discrete range of 3 values. The range is discrete and there is no order between the values.
  - *ordinal properties* such as `rating` with possible range values 1star - 5star. The range is discrete and the range values are ordered.
- Numerical Properties:
  - *discrete properties* such as `noLikes`. The range is usually infinite usually an infinite subset of integers (distinct and separated values).
  - *continuous properties* such as `visitLength`. The range is usually infinite usually occupying an interval of real numbers.

3.1.1 Organization, Place

The class `Organization` describes an organization from the perspective of its location members and digital channels it manages. Obvious properties such as `name` or `image` are part of its description too. In Metamarket organizations are owners of digital content (see the abstract class Content). The organization members (Contributor) are content creators. Particularly to Metamarket, organizations are in the domain of the property `channels` featuring the online presence marketing channels an organization manages (Fig. 10).

Example 1 (*Organization*)

```
<http://metamarket.info/org/12345> [
  rdf:type <http://schema.org/Organization>;

  b:websites <http://metamarket.info/domains/12345>;
  b:websites <...>;
```

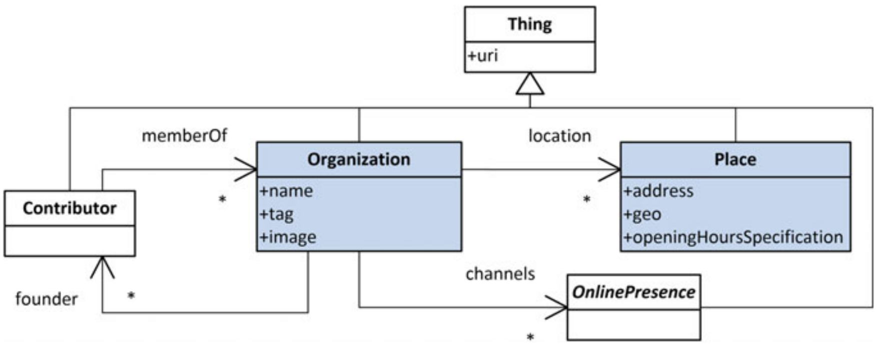


Fig. 10 Metamarket organization model

```
    b:individualApps <http://metamarket.info/apps/12345>;
    b:individualApps <...>;

    b:communityApps <http://metamarket.info/apps/12346>;
    b:communityApps <...>;

    b:tag "Restaurant"@de;
    b:tag "...";

    s:location <http://metamarket.info/place/c30726a4>;
    s:founder <http://metamarket.info/person/c212dc21>
  ]
```

`Place` is a class used for describing a location. This is class is part of the domain of various location related properties such as `address` or `geo`. In Metamarket `Place` is also in the domain of `openingHoursSpecification` as described in the below example.

Example 2 (*Place*)

```
  <http://metamarket.info/place/24309384> [
    rdf:type <http://schema.org/Place>;

    s:address [
      s:type <http://schema.org/PostalAddress>;
      s:streetAddress "Berliner Platz 6";
      s:postalCode "03046";
      s:addressLocality "Cottbus";
      s:addressCountry "Germany";
    ];

    s:geo [
      rdf:type <http://schema.org/GeoCoordinates>;
      s:elevation "100.23"^^xs:float; // actually not set
      s:latitude "50.23442"^^xs:float;
      s:longitude "45.0000"^^xs:float;
    ];

    s:openingHoursSpecification [
      s:type <http://schema.org/OpeningHoursSpecification>;
      s:closes "19:00:00"^^xs:time;
      s:opens "09:00:00"^^xs:time;
      s:dayOfWeek "Monday";
    ];

    s:openingHoursSpecification [
      s:type <http://schema.org/OpeningHoursSpecification>;
      s:closes "19:00:00"^^xs:time;
      s:opens "09:00:00"^^xs:time;
      s:dayOfWeek "Tuesday";
    ];
  ].
```

3.1.2 Persona

Metamarket employs a consumer-centered design. The class `Persona` describes the fictional visitors representing different user types that use one or more online presence channels to interact with the content provided by organizations. Metamarket personas are described by properties such as `ageGropup`, `gender` and `interest`. Due to the open and modular character of the Metamarket ontology, `Persona` can be extended with other kinds of properties according with users use cases (Fig. 11).

Metamarket Persona is suppose to build on user sharing of data and as well as inferences based on users activities. We consider certainty factors model to infer personas from behavioral patterns.

The current persona profile is based on the below properties.

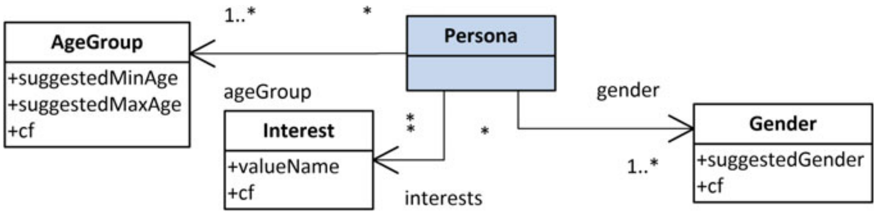


Fig. 11 Metamarket persona model

- `ageGroup` is a property with 7 possible values: `13-20`, `21-30`, `31-40`, `41-50`, `51-60`, `61-70`, and `>71`. Minimum 1 value is required (cannot be empty)
- `interest` is a property with at most 5 values (keywords). Min 1 value is required (cannot be empty)
- `gender` is a property with 3 possible values. Min 1 value is required (cannot be empty)
- `cf` is the confidence factor. `cf="1.0"8s:float` means complete confidence. If cf is missing then the default value is 1.

Example 3 (*Persona*)

```
<http://metamarket.info/person/3465754667> [
  rdf:type <http://schema.org/Person>;

  s:url "http://facebook..."^^xs:anyURI; // private.

  b:informs _:b39220121x61717;

  b:profile [
    rdf:type <http://metamarket.info/Persona>;

     b:platform "Android";
     b:version "4.1.2";
     b:gender : [
       rdf:type <http://metamarket.info/GenderType>;
       b:suggestedGender": "male";
       b:cf "0.6"^^xs:float;
     ];
     b:ageGroup  [
      rdf:type <http://metamarket.info/AgeGroup>;
      b:suggestedMinAge "31"^^xs:positiveInteger;
      b:suggestedMaxAge "40"^^xs:positiveInteger;
     ];
     b:interest [
        rdf:type <http://metamarket.info/Interest>;
        b:valueName "education"@en;
        b:cf "0.6^^xs:float
     ];
     b:interest [
        rdf:type <http://metamarket.info/Interest>;
        b:valueName "lifestyle"@en;
        b:cf "0.4^^xs:float
     ];

  ];
]
```

3.1.3 Online Presence

Online Presence is the sum of all digital platforms that can be used by an individual or a company to present and draw attention on services and products they provide. Nowadays the most important online communication channels where you can meet prospects and customers are:

1. *The Mobile Channel*: the organization mobile application, community mobile apps etc.
2. *The Social Media Channel*: Facebook, Twitter, LinkedIn, Xing, Instagram, Google+, private social networks.
3. *The Web Channel*: the organization website - world wide accessible, multi language, indexed by search engines.

Example 4 (*Online Presence*)

```
<http://metamarket.info/domain/c212dc21> [
  rdf:type <http://schema.org/WebPage>;
  b:id "c212dc21-4670-4206-a569-a0e43b7e6990"; // private
  s:name "some.domain.de"; // private
].

<http://metamarket.info/app/c212dc22> [
  rdf:type <http://schema.org/MobileApplication>;
  s:name "App name"; // private
  b:appType "individual"; // "community"

  b:playStoreUrl "https://..."^^xs:anyURI; // private
  b:appStoreUrl "https://..."^^xs:anyURI; // private
  //...
].
```

3.1.4 Article and Subclasses

Following the development of Schema.org creative content, Metamarket introduces a basic set of news content:

1. `Article` – content general interest or of a specific topic.
2. `BlogPosting` – news content usually with relatively short life time. An announcement the organization is doing to its followers.
3. `Poster` – a meaningful and representative picture of an organization. It may feature a product an achievement, a landscape and so on.
4. `Video` – a video of an organization published on online presence channels. Usually it had a title and a description and a media object encoding the video

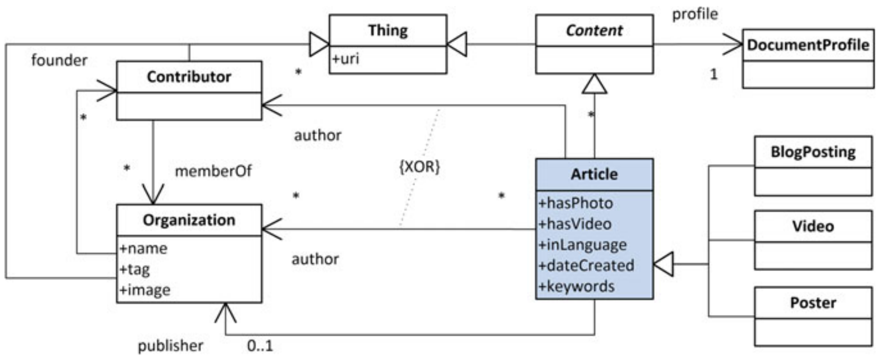


Fig. 12 Metamarket article model

content. In addition the creator may provide other information such as language information and keywords (Fig. 12).

Example 5 (*Article*)

```
<http://metamarket.info/post/3842852384> [
  rdf:type <http://schema.org/BlogPosting>;

  b:hasPhoto "true"^^xs:boolean;
  b:hasVideo "false"^^xs:boolean;

  s:url "http://..."^^xs:anyURI; //private
  s:name "Release of Deliberation RuleML 1.01"; //private
  s:headline "The specification ..."; //private

  s:inLanguage "en-GB";
  s:dateCreated "2014-07-30T00:54:25+02:00"^^xs:dateTime;
  s:author <http://metamarket.info/org/73201e52>;
  s:publisher <http://metamarket.info/org/73201e52>;
  s:keywords "Version Release";
  s:keywords "Deliberation RuleML";

  b:offers <http://metamarket.info/offer/464564687>;

  b:profile [
    rdf:type <http:http://metamarket.info/DocumentProfile>;
    ...
  ];
].
```

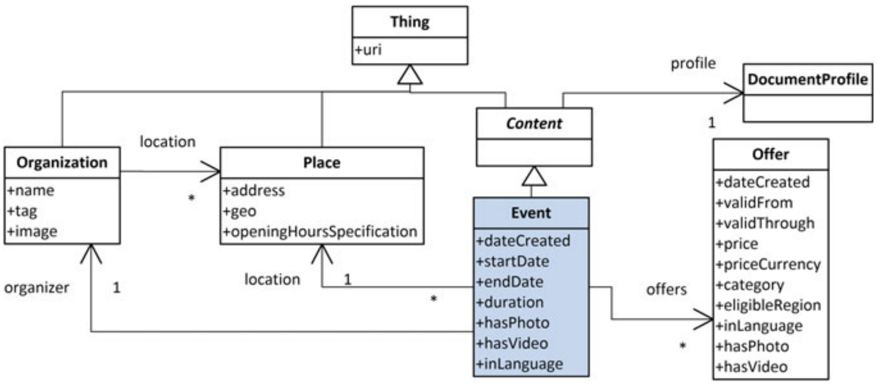


Fig. 13 Metamarket event model

3.1.5 Event

The class Event is defined to describe various real life events such as festivals, an artist performing on stage, media events created for advertising, party, sport events such as championship and so on. All of these share a temporal availability encoded by properties such as `startDate` `endDate`, `duration`.

Metamarket events may also feature `offers` such as ticket sales or brand object sales. Obviously an application may reason on the temporal properties – an event may *happen in the future*, *close to actual time*, happening *right now* or *expired* (happened in the past) (Fig. 13).

Example 6 (*Event*)

```
<http://metamarket.info/event/8289247> [
  rdf:type <http://schema.org/Event>;

  s:url "http://..."^^xs:anyURI;//private

  s:organizer <http://metamarket.info/org/23227327>;
  s:dateCreated "2012-10-01T14:00:00+01:00"^^xs:dateTime;
  s:startDate "2012-11-06T00:00:00+01:00"^^xs:dateTime;
  s:endDate "2012-11-11T23:59:59+01:00"^^xs:dateTime;
  s:duration "P5D"^^xs:duration;
  s:location <http://metamarket.info/place/24309384>;
  s:offers <http://metamarket.info/offer/464564687>;
  s:offers <http://metamarket.info/offer/464535456>;

  b:hasPhoto "true"^^xs:boolean;
  b:hasVideo "true"^^xs:boolean;
```

```
    b:inLanguage "de-DE";
    b:summary "..."; //private
    b:profile [
      rdf:type <http:http://metamarket.info/DocumentProfile>;
      ...
    ];
  ].
```

3.1.6 Offer

The class Offer describes an organization product or service offered to the potential customers at a specific `price` and during a limited amount of time (`validFrom`, `validThrough`). Additional properties such as `seller`, `eligibleRegion`, as well as language information can be provided (Fig. 14).

Example 7 (*Offer*)

```
  <http://metamarket.info/offer/464564687> [
    rdf:type <http://schema.org/Offer>;

    s:url "http://..."^^xs:anyURI; //private
    s:seller <http://metamarket.info/org/9a303878>;
    s:description "The offer description"; //private
    s:validFrom "2012-11-01T23:59:59+01:00"^^xs:dateTime;
    s:validThrough "2012-11-02T23:59:59+01:00"^^xs:dateTime;
    s:price "49.00"^^xs:float;
    s:priceCurrency "EUR";
    s:category "film";
    s:category "other";
    s:eligibleRegion "Lausitz";
    s:offers <http://metamarket.info/offer/464535459>;

    b:inLanguage "de-DE";
    b:dateCreated "2014-05-07T10:45:17+02:00"^^xs:dateTime;

    b:hasPhoto "true"^^xs:boolean;
    b:hasVideo "false"^^xs:boolean;
    b:profile [
      rdf:type <http:http://metamarket.info/DocumentProfile>;
      ...
    ];
  ].
```

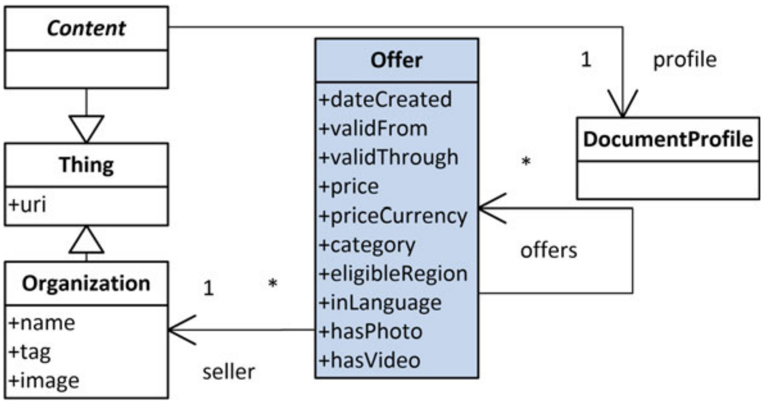


Fig. 14 Metamarket offer model

3.1.7 Coupon

Metamakret `Coupon` features a special kind of offer particularly from the perspective of `price` formation and availability. A coupon can be issued at a fixed price, limited quantity (`inventoryLevel`) or the price can evolve according with the seller auction rules (`priceAuction`), or any consistent combination of these properties (Fig. 15).

Example 8 (*Coupon*)

```
<http://metamarket.info/coupon/454664687> [
  rdf:type <http://metamarket.info/Coupon>;

  b:id "564687464"; //private
  s:name "FestPass"; //private
  s:url "http://..."^^xs:anyURI; //private
  s:description "some description"; //  private
  s:image ""http://..."^^xs:anyURI"//private.
  s:seller <http://metamarket.info/org/9a303878>;

  b:dateCreated "2016-07-07T10:45:17+02:00"^^xs:dateTime;

  s:validFrom "2012-11-01T23:59:59+01:00"^^xs:dateTime;
  s:validThrough "2012-11-02T23:59:59+01:00"^^xs:dateTime;

  s:price "49.00"^^xs:float; // may have a price
  s:priceCurrency "EUR";

  s:inventoryLevel [
    rdf:type <http://schema.org/QuantitativeValue>;
    s:value "101"^^xs:positiveInteger;
  ];

  b:priceAuction [
    rdf:type <http://metamarket.info/PriceAuction>;
    b:priceAuctionSpecification
      <http://metamarket.info/PriceAuction/1234sli45ys4687>
  ];

  b:priceAuction [
    rdf:type <http://metamarket.info/PriceAuction>;
    b:priceAuctionSpecification
      <http://metamarket.info/PriceAuction/1234sli45ys4688>
  ];


  s:category "pants";
  s:eligibleRegion "Berlin";

  b:inLanguage "de-DE";

  b:hasPhoto "true"^^xs:boolean;
  b:hasVideo "true"^^xs:boolean;
  b:profile [
    rdf:type <http:http://metamarket.info/DocumentProfile>;
    ...
  ];
].
```

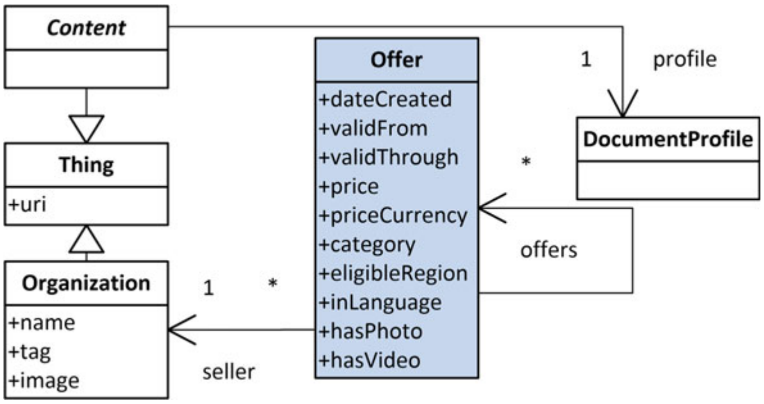


Fig. 15 Metamarket coupon model

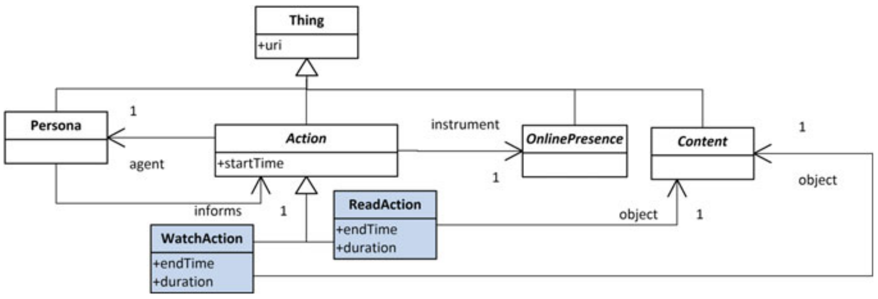


Fig. 16 Metamarket ReadAction, WatchAction model

3.2 Metamarket Actions

Metamarket actions are events produced by an actor (`Persona`) using a specific channel (`instrument`) on a specific content (see `Content`). Some of the actions are *duration events* i.e., they define `startTime`, `endTime` and duration values. Apart of the ontological definition a Metamarket application may define supplementary properties such as `referrer` to be used to better understand the user behavior when navigating on organization pages.

3.2.1 ReadAction, WatchAction

A `ReadAction` is a duration event and happens when a visitor reads the content produced by an organization on a specific online presence channel. For example, a visitor reading a blog post published in a mobile application, on a Facebook page or on a website. `ReadAction` is the act of consuming written content (Fig. 16).

A `WatchAction` is a duration event happening when a visitor watch visual content published by an organization. The visual content can be placed in any of the specialized kinds of content (`Article`, `BlogPosting`, `Poster`, `Video`, `Event`, `Offer`, `Coupon`). `WatchAction` is the act of consuming visual content.

Example 9 (*ReadAction*)

```
<http://metamarket.info/action/1s2ffgvb4309fe4> [
  rdf:type <http://schema.org/ReadAction>;
  s:agent <http://metamarket.info/person/24309384>;
  s:object <http://metamarket.info/article/26967ca6>;
  s:instrument [
    rdf:type <http://schema.org/MobileApplication>
    b:id "UUID"// private
  ];
```

```
    s:startTime "2012-10-01T14:00:00+01:00"^^xs:dateTime;
    s:endTime "2012-10-01T14:05:00+01:00"^^xs:dateTime;
    b:duration "P00Y00M00DT00H05M00S"^^xs:duration;

    b:referrer "http://..."^^xs:anyURI; // private
   ].
```

3.2.2 CommunicateAction, SMSAction

A `CommunicateAction` happens when a persona have a phone call with an organization using one of the online presence channels. Frequently this happens using a mobile application (the `instrument` type is `"http://schema.org/MobileApplication"`) and typically this action is recorded when the *user intends to start the call*. As such, the `CommunicateAction` while defines a `startTime` does not define a an `endTime` and, by consequence a `duration` (Fig. 17).

A `SMSAction` is very similar with `CommunicateAction` but encodes the fact that the communication act is not voice but text.

Example 10 (*CommunicateAction*)

```
  <http://metamarket.info/action/1s2ffgvb4309384> [
    rdf:type <http://schema.org/CommunicateAction>;
    s:agent <http://metamarket.info/person/24309384>;
    s:object <http://metamarket.info/place/20894weer0031>;
    s:instrument [
      rdf:type <http://schema.org/MobileApplication>
      b:id "UUID"
    ];
    s:startTime "2012-10-01T14:00:00+01:00"^^xs:dateTime;
   ].


  <http://metamarket.info/action/1s2ffgvb4309aa4> [
    rdf:type <http://schema.org/CommunicateAction/SMSAction>;
    ...
   ].
```

3.2.3 MoveAction

A `MoveAction` happens when an agent uses a specific instrument to (virtually) navigate to a specified location. Apart of the usual `agent`, and `instrument`

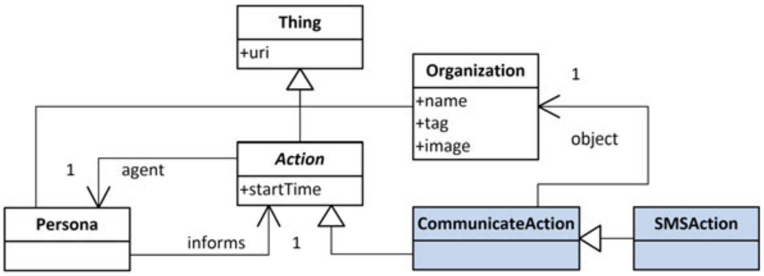


Fig. 17 Metamarket CommunicateAction, SMSAction model

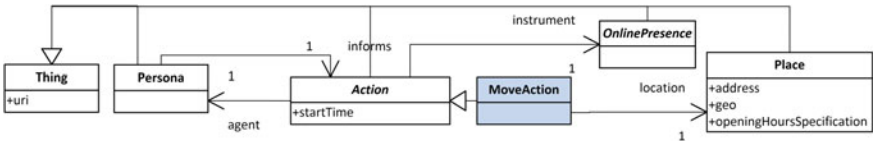


Fig. 18 Metamarket MoveAction model

such action defines the property `toLocation` (with range `Place`) meaning the encoding of the location were the agent intends to relocate (Fig. 18).

Example 11 (*MoveAction*)

```
<http://metamarket.info/action/2a2fae23b4309aa4> [
  rdf:type <http://schema.org/MoveAction>;
  s:agent <http://metamarket.info/person/24309384>;
  s:toLocation <http://metamarket.info/place/20894weer0031>;
  s:instrument [
    rdf:type <http://schema.org/WebApplication>
    s:url "http://..."^^xs:anyURI //private
  ];
  s:startTime "2012-10-01T14:00:00+01:00"^^xs:dateTime;
 ].
```

3.2.4 OrganizeAction

Schema.org defines such action as “The act of manipulating, administering, supervising or controlling one or more objects.”. Metamarket uses `OrganizeAction` to record the event of an agent saving an organization business card into his business cards list (Fig. 19).

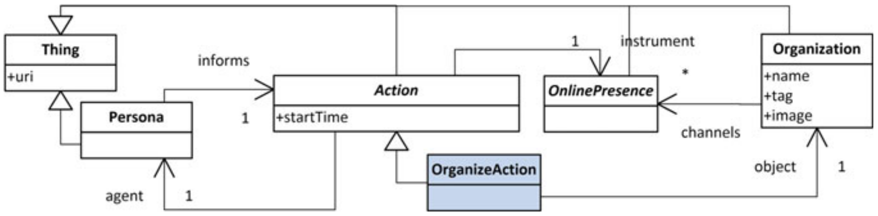


Fig. 19 Metamarket OrganizeAction model

Example 12 (*OrganizeAction*)

```
<http://metamarket.info/action/2a2fsd2234357aae> [
  rdf:type <http://schema.org/OrganizeAction>;
  s:agent <http://metamarket.info/person/24309384>;
  s:object <http://metamarket.info/org/1335466097>;
  s:instrument [
    rdf:type <http://schema.org/MobileApplication>
    b:id "UUID" // private
  ];
  s:startTime "2012-10-01T14:00:00+01:00"^^xs:dateTime;
 ].
```

3.2.5 ScheduleAction, JoinAction

A `ScheduleAction` happens when a persona intends to schedule one or more tasks. Particularly this can be an "add to calendar" action i.e. the act of a person saving to his own calendar a specific event (Fig. 20).

A `JoinAction` happens when an agent express its intent to join a specific event published by an organization on a specific channel.

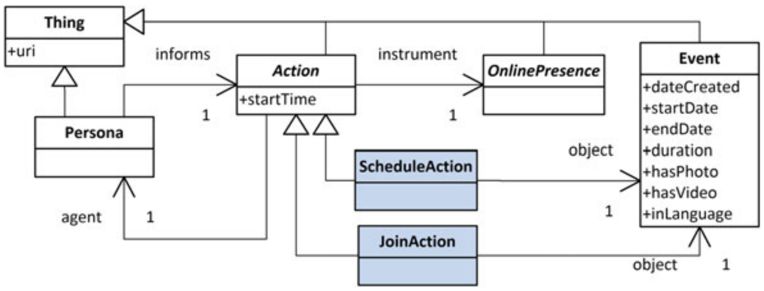


Fig. 20 Metamarket ScheduleAction, JoinAction models

Both actions does not have duration as they are modeled as intents and therefore not monitored.

Example 13 (*Schedule, Join*)

```
<http://metamarket.info/action/2a2fsd2234aer47e> [
  rdf:type <http://schema.org/ScheduleAction>;
  s:agent <http://metamarket.info/person/24309384>;
  s:object <http://metamarket.info/event/208940031>;
  s:instrument [
    rdf:type <http://schema.org/WebApplication>
    s:url "http://..."^^xs:anyURI //private
  ];
  s:startTime "2012-10-01T14:00:00+01:00"^^xs:dateTime;
 ].

<http://metamarket.info/action/2a2fsd791e3f7baf> [
  rdf:type <http://schema.org/JoinAction>;
  ...
 ].
```

3.2.6 ConsumeAction

A `ConsumeAction` happens when an actor claim an `Offer` or consume a `Coupon`. As for the other defined actions, Metamarket does not make any assumption of the application layer workflow implementation. Each application using Metamarket should provide its own workflow implementation (Fig. 21).

Example 14 (*ConsumeAction*)

```
<http://metamarket.info/action/asr45nd7d58cfn58> [
  rdf:type <http://schema.org/ConsumeAction>;
  s:agent <http://metamarket.info/person/24309384>;
  s:object <http://metamarket.info/offer/178a1a1f>;
  s:instrument [
    rdf:type <http://schema.org/MobileApplication>
    b:id "UUID"
  ];
  s:startTime "2012-10-01T14:00:00+01:00"^^xs:dateTime;
 ].
```

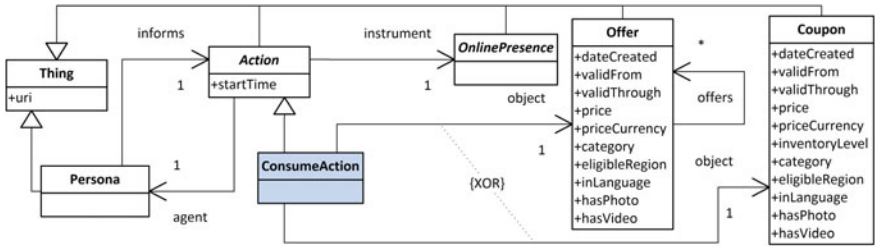


Fig. 21 Metamarket ConsumeAction model

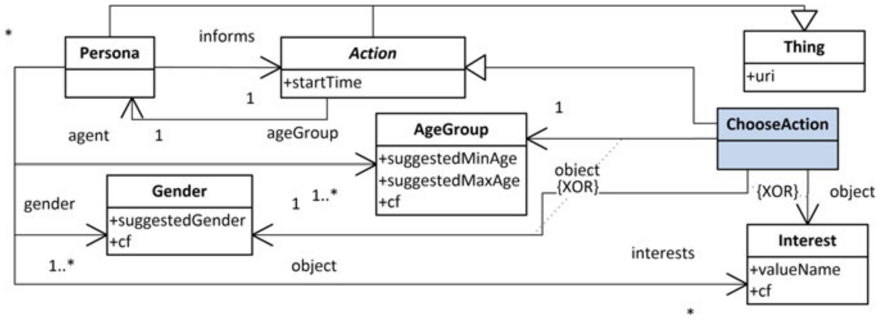


Fig. 22 Metamarket ChooseAction model

3.2.7 ChooseAction

A ChooseAction occurs when a actor agree to set values for its Persona profile. As in the below example, there may be many objects produced by this action such as one for age group profiling or about user's interests (Fig. 22).

Example 15 (*ChooseAction*)

```
<http://metamarket.info/action/a3sh45dsj74gt5> [
  rdf:type <http://schema.org/ChooseAction>;
  s:agent <http://metamarket.info/person/24309384>;
  s:object [
    rdf:type <http://schema.org/PeopleAudience>;
    s:name "31-40"; //private
    s:suggestedMinAge 31;
    s:suggestedMaxAge 40;
    s:suggestedGender "male"@en;

  ];
  s:object [
    rdf:type <http://schema.org/Intangible/PropertyValueSpecification>;
    s:multipleValues <http://schema.org/True>;
    s:name "Interest"@en; //private
    s:valueName "education"@en;
    s:valueName "lifestyle"@en;
  ];
  s:instrument [
    rdf:type <http://schema.org/MobileApplication>
    b:id "UUID"
  ];
  s:startTime "2012-10-01T14:00:00+01:00"^^xs:dateTime;
 ].
```

3.2.8 TrackAction

A `TrackAction` occurs when a consumer request content available only for a specific location. So, is not the application requesting the user location. Location is delivered either the request comes from a mobile presence or from a web application. The action also encodes the request method: either an usual text search or by consumer providing his own location. A radius related to the center of the location can be provided too (Fig. 23).

Example 16 (TrackAction)

```
  <http://metamarket.info/action/a5dkf74nf3> [
    rdf:type <http://schema.org/TrackAction>;
    s:agent <http://metamarket.info/person/24309384>;
    s:instrument [
      rdf:type <http://schema.org/WebApplication>
      s:url "http://binarypark.org"^^xs:anyURI
    ];
    s:deliveryMethod
         <http://metamarket.info/deliveryMethod/textSearch> ;
         // <http://metamarket.info/deliveryMethod/geoLocation>
    s:startTime "2015-10-01T14:00:00+01:00"^^xs:dateTime;
    s:result [
      rdf:type <http://schema.org/GeoCircle>
      s:geoMidpoint [
        rdf:type <http://schema.org/GeoCoordinates> ;
        s:latitude "40.75";
        s:longitude "73.98";
        s:address [
          rdf:type <https://schema.org/PostalAddress> ;
          s:addressCountry "de";
          s:addressLocality "Berlin";
          s:addressRegion "Brandenburg";
        ]
      ];
      s:geoRadius <http://metamarket.info/radius/km25> // km50, km100
    ]
  ].
```

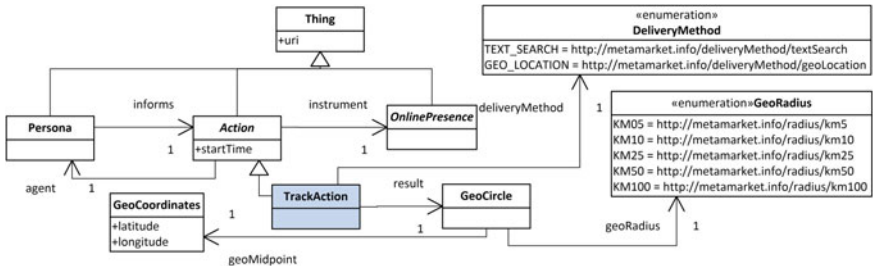

Thing
+uri
«enumeration»
DeliveryMethod
TEXT_SEARCH = http://metamarket.info/deliveryMethod/textSearch
GEO_LOCATION = http://metamarket.info/deliveryMethod/geoLocation
informs
instrument
deliveryMethod
«enumeration»GeoRadius
KM05 = http://metamarket.info/radius/km5
KM10 = http://metamarket.info/radius/km10
KM25 = http://metamarket.info/radius/km25
KM50 = http://metamarket.info/radius/km50
KM100 = http://metamarket.info/radius/km100
Persona
Action
+startTime
OnlinePresence
agent
result
TrackAction
GeoCircle
GeoCoordinates
+latitude
+longitude
geoRadius
geoMidpoint

Fig. 23 Metamarket TrackAction model

3.2.9 CommentAction

The `CommentAction` records the act of generating a comment about a published content. `OnlinePresence` channels such as `Website`, `App`, `Facebook`, `Twitter`, `LinkedIn`, `Xing`, `Google+` allows readers to react to the content in the form of comments (Fig. 24).

Example 17 (*CommentAction*)

```
<http://metamarket.info/action/t4llo56dndd64> [
  rdf:type <http://schema.org/CommentAction>;
  s:agent <http://metamarket.info/person/24309384>;
  s:object <http://metamarket.info/article/26967ca6>;
  s:instrument [
    rdf:type <http://schema.org/MobileApplication>
             //  <http://schema.org/WebApplication>
    b:id "UUID"
    // s:url "http://binarypark.org"^^xs:anyURI
  ];
  s:startTime "2012-10-01T14:00:00+01:00"^^xs:dateTime;
 ].
```

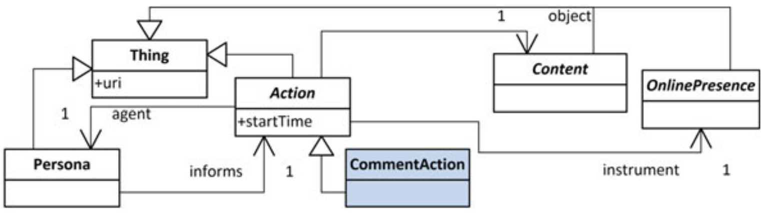


Fig. 24 Metamarket CommentAction model

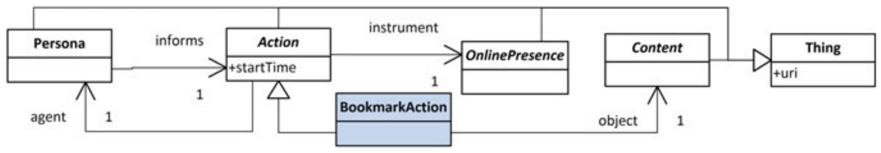


Fig. 25 Metamarket BookmarkAction model

3.2.10 BookmarkAction

The `BookmarkAction` is the action of an agent to mark a specific content published by an organization on one of its online presence channels. For example on Facebook an user can *"Save a post"*, on an e-commerce website an user can add *a product to the wishlist* and so on (Fig. 25).

Example 18 (*BookmarkAction*)

```
<http://metamarket.info/action/ag57dk6h73> [
  rdf:type <http://schema.org/BookmarkAction>;
  s:agent <http://metamarket.info/person/24309384>;
  s:object <http://metamarket.info/event/133546335460973354>;
  s:instrument [
    rdf:type <http://schema.org/MobileApplication>
    b:id "UUID"
  ];
  s:startTime "2012-10-01T14:00:00+01:00"^^xs:dateTime;
 ].
```

3.2.11 VoteAction

The `VoteAction` is the action of an agent to rate specific content published by an organization on one of its online presence channels. On concrete channels rating takes various forms. By default Metamarket uses a *five star rating scale* (rating values from `1` to `5`) but an application may define another object as result of this action (Fig. 26).

Voting may be used by a third party application to asses consumer preferences [1].

Example 19 (*VoteAction*)

```
<http://metamarket.info/action/2r456dnt65> [
  rdf:type <http://schema.org/VoteAction>;
  s:agent <http://metamarket.info/person/24309384>;
  s:object <http://metamarket.info/article/26967ca6-01928>;
  s:instrument [
    rdf:type  <http://schema.org/WebApplication>
    s:url "http://binarypark.org"^^xs:anyURI
  ];
  s:startTime "2012-10-01T14:00:00+01:00"^^xs:dateTime;
  s:option [
     rdf:type <http://schema.org/Rating>;
     s:ratingValue "4"
  ];
 ].
```

3.2.12 LikeAction, DislikeAction, ShareAction

The `LikeAction` (`DislikeAction`) encodes the agent sentiment against the content published on a specific online channel such as Facebook or Twitter, but, of course it can be implemented on websites and apps too. Like and dislike may be used, together with voting to asses consumer preferences [1] (Figs. 27, 28).

The `ShareAction` is the act of distribution content to other people. It includes email and typical sharing on social media.

Example 20 (*LikeAction*)

```
  <http://metamarket.info/action/asd4g56b90> [
    rdf:type <http://schema.org/LikeAction>;
            // <http://schema.org/DislikeAction>;
            // <http://schema.org/ShareAction>;
    s:agent <http://metamarket.info/person/24309384>;
    s:object <http://metamarket.info/offer/464564687>;
    s:instrument [
      rdf:type <http://schema.org/MobileApplication>
      b:id "UUID"
    ];
    s:startTime "2012-10-01T14:00:00+01:00"^^xs:dateTime;
   ].
```

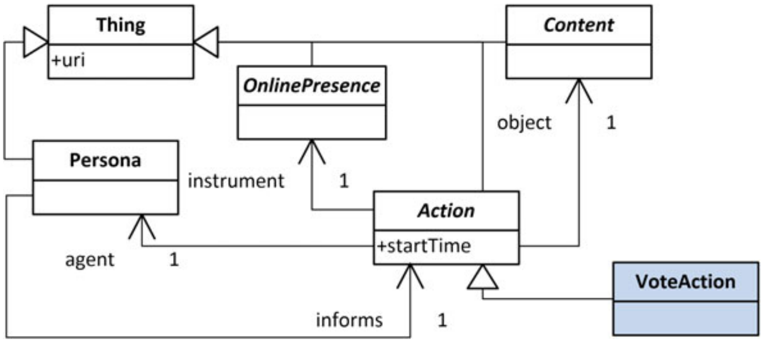


Fig. 26 Metamarket VoteAction model

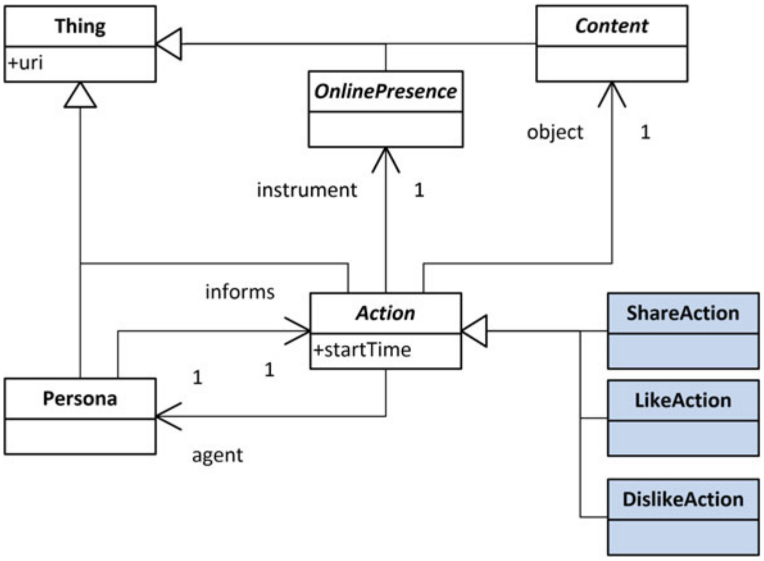


Fig. 27 Metamarket LikeAction, DislikeAction, ShareAction models

3.2.13 RegisterAction, UnRegisterAction, ViewAction

Example 21 (*RegisterAction*)

```
<http://metamarket.info/action/ar3n35dnd53> [
  rdf:type <http://schema.org/RegisterAction>;
  s:agent <http://metamarket.info/person/24309384>;
  s:object <http://metamarket.info/org/208940031>;
  s:instrument [
    rdf:type <http://schema.org/MobileApplication>
    //  <http://schema.org/WebApplication>
    b:id "UUID"
    // s:url "http://binarypark.org"^^xs:anyURI
  ];
  s:startTime "2014-10-01T14:00:00+01:00"^^xs:dateTime;
 ].
```

4 Final Remarks

Ontologies are becoming a recognized vehicle for knowledge reuse, knowledge sharing, and modeling. In this chapter we presented Metamarket, a practical ontology of actions for the description of user interactions on the digital world. The relevance of

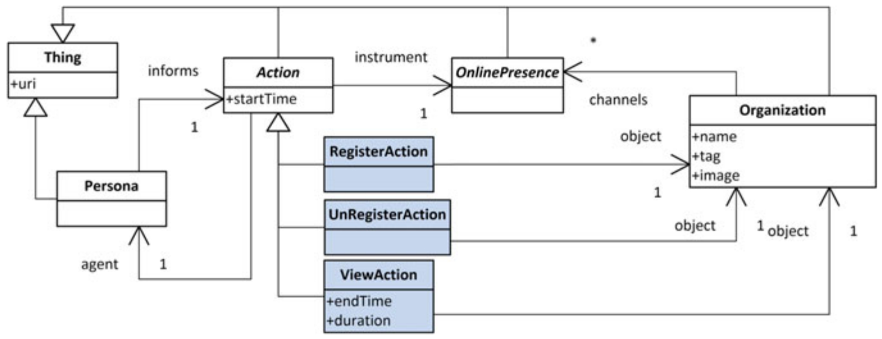


Fig. 28 Metamarket RegisterAction, UnRegisterAction, ViewAction models

this work is motivated by the difficulty that data analyzers have in determining which vocabularies to use to describe user interactions with the digital content. The main results described in this chapter include: (1) a state of the art on research on activity ontologies, (2) the complete UML model of the Metamarket ontology including examples for each of the significant concepts and (3) availability of this ontology at http://www.metamerket.info ready to be used by other practitioners. The ontology is implemented using the Web Ontology Language (OWL) standard. Metamarket can be used to develop intelligent UX applications, responsive to user needs. Other interesting future work refers to extend Metamarket towards describing and inferring emotions. The first step should concern the description of user affective states. Metamarket can be easily aligned with Schema.org vocabulary, and give the user the complete freedom with respect of extensibility. In addition, enterprise applications would have great benefits using Metamarket particularly in areas such as content generation related to the user activities.

Acknowledgements We would like to express our very great appreciation to Prof. Daniel Baier from Universität Bayreuth for his valuable and constructive suggestions during the planning of this research work. Particularly, his technical suggestions with respect of marketing research requirements for deriving preferences are very much appreciated. We also want to express our gratitude to Matthias Tylkowski and Martin Müller from Binarypark, the company hosting and maintaining all our running services.

References

1. Giurca, A., Baier, D., Schmitt, I.: What is in a like? Preference aggregation on the social web. Data Science, Learning by Latent Structures, and Knowledge Discovery, pp. 435–444. Springer, Berlin (2015)
2. Meditskos, G., Dasiopoulou, S., Efstathiou, V., Kompatsiaris, I.: Ontology patterns for complex activity modelling. In: International Workshop on Rules and Rule Markup Languages for the Semantic Web, pp. 144–157. Springer, Berlin (2013)

3. Suarez-Figueroa, M.C., Gomez-Perez, A.: Towards a glossary of activities in the ontology engineering field. In: Proceedings of the Sixth International Language Resources and Evaluation (LREC'08), Marrakech, Morocco, 28–30 May 2008
4. Dewabharata, A., Wen, D.M.H., Chou, S.Y.: An activity ontology for context-aware health promotion application. In: 2013 IEEE 37th Annual Computer Software and Applications Conference Workshops (COMPSACW), pp. 421–426. IEEE (2013)
5. Thakker, D., Denaux, R., Dimitrova, V.: Activity model ONtology (AMOn) (2011). http://imash.leeds.ac.uk/ontology/amon/
6. Ferreira, D.R., Alves, S., Thom, L.H.: Ontology-based discovery of workflow activity patterns. In: International Conference on Business Process Management, pp. 314–325. Springer, Berlin (2011)
7. Gruninger, M., Katsumi, M.: An activity-based ontology for dates. In: 2015 AAAI Spring Symposium Series (2015)
8. Chen, L., Nugent, C.: Ontology-based activity recognition in intelligent pervasive environments. Int. J. Web Inf. Syst. **5**(4), 410–430 (2009)
9. Abdalla, A., Hu, Y., Carral, D., Li, N., Janowicz, K.: An ontology design pattern for activity reasoning. In: Proceedings of the 5th International Conference on Ontology and Semantic Web Patterns, vol. 1302, pp. 78–81 (2014). http://CEUR-WS.org
10. Konstantinos, A., Ioannis, R.: Activity ontologies for intelligent calendar applications. In: Proceedings of the 7th Balkan Conference on Informatics Conference, p. 17. ACM (2015)
11. Troncy, R., Malocha, B., Fialho, A.T.: Linking events with media. In: Proceedings of the 6th International Conference on Semantic Systems, p. 42. ACM (2010). Ontology available at http://linkedevents.org/ontology/
12. Riboni, D., Bettini, C.: OWL 2 modeling and reasoning with complex human activities. Pervasive Mob. Comput. **7**(3), 379–395 (2011)
13. Morgenstern, L., Riecken, D.: SNAP: an action-based ontology for e-commerce reasoning. In: Formal Ontologies Meet Industry, Proceedings of the 1st International Workshop FOMI (2005). http://www-formal.stanford.edu/leora/snap9z.pdf. Accessed 15 Aug 2015
14. Brickley, D., Miller, L.: FOAF vocabulary specification 0.99. Namespace document (2012). http://xmlns.com/foaf/spec/
15. Hobbs, J.R., Pan, F.: An ontology of time for the semantic web. ACM Trans. Asian Lang. Inf. Proces. (TALIP) **3**(1), 66–85 (2004)
16. Lieberman, J., Singh, R., Goad, C.: W3C geospatial vocabulary. W3C Incubator Group Report (2007)
17. Shaw, R., Troncy, R., Hardman, L.: LODE: linking open descriptions of events. In: 4th Asian Semantic Web Conference (ASWC'09) (2009)
18. Van Hage, W.R., Malaise, V., Segers, R., Hollink, L., Schreiber, G.: Design and use of the simple event model (SEM). Web Sem.: Sci. Serv. Agents World Wide Web **9**(2), 128–136 (2011)
19. Guizzardi, G., Wagner, G.: Towards an ontological foundation of discrete event simulation. In: Johansson, B., Jain, S., Montoya-Torres, J., Hugan, J., Yücesan, E. (eds.) Proceedings of Winter Simulation Conference, pp. 652–664. IEEE, Baltimore (2010)
20. Guizzardi, G., Wagner, G.: Towards an ontological foundation of agent-based simulation. In: Jain, S., Creasey, R.R., Himmelspach, J., White, K.P., Fu, M. (eds.) Proceedings of Winter Simulation Conference. IEEE, Phoenix (2011)
21. Knox, S., Coyle, L., Dobson, S.: Using ontologies in case-based activity recognition. In: Proceedings of the Twenty-Third International Florida Artificial Intelligence Research Society Conference (FLAIRS'10), Daytona Beach, Florida, 19–21 May 2010
22. Snell, J., Prodromou, E.: Activity vocabulary W3C working draft 31 May 2016. https://www.w3.org/TR/activitystreams-vocabulary/

OntoMaven - Maven-Based Ontology Development and Management of Distributed Ontology Repositories

Adrian Paschke and Ralph Schäfermeier

Abstract In collaborative agile ontology development projects support for modular reuse of ontologies from large existing remote repositories, ontology project life cycle management, and transitive dependency management are important needs. The Apache Maven approach has proven its success in distributed collaborative Software Engineering by its widespread adoption. The contribution of this paper is a new design artifact called OntoMaven. OntoMaven adopts the Maven-based development methodology and adapts its concepts to knowledge engineering for Maven-based ontology development and management of ontology artifacts in distributed ontology repositories.

1 Introduction

Sharing and reusing knowledge in ontology-based applications is one of the main aims in the Corporate Semantic Web[1] as well as the Pragmatic Web[2] [1–3], which requires the support of distributed ontology management, documentation, validation and testing. As pointed out in [4, 5] such ontology development life cycles have a similar structural and logic complexity as distributed software development projects. Agile ontology life cycle management methodologies, such as COLM[3] [6], require the collaboration of knowledge engineers and domain experts. Ontologies are developed and maintained in an iterative and distributed way, which requires the support of versioning [7, 8] and modularization [9, 10]. Moreover, new aspect-oriented

[1]http://www.corporate-semantic-web.de.
[2]http://www.pragmaticweb.info.
[3]http://www.corporate-semantic-web.de/colm.html.

A. Paschke (✉) · R. Schäfermeier
Computer Science Institute, Corporate Semantic Web Group,
Freie Universität Berlin, Königin-Luise-Str. 24/26, 14195 Berlin, Germany
e-mail: paschke@inf.fu-berlin.de

R. Schäfermeier
e-mail: ralph.schafermeier@gmail.com

G.J. Nalepa and J. Baumeister (eds.), *Synergies Between Knowledge Engineering and Software Engineering*, Advances in Intelligent Systems and Computing 626, https://doi.org/10.1007/978-3-319-64161-4_12

ontology development approaches [11] enable weaving of cross-cutting knowledge concerns into the main ontology model, which requires meta-level descriptions of ontology aspects and management of distributed knowledge models.

In this work we adapt a highly successful method and tool in distributed software engineering project management, namely Apache Maven,[4] for the Maven-based management of distributed ontology repositories. Maven is a tool for the (distributed) project management and quality assurance in software engineering projects. The goal of Maven is to automate recurring tasks in software development projects such as management of software artifact in distributed remote and local repositories, versioning, dependency management, documentation, testing, packaging and deployment.

We follow a Design Science research methodology and develop a novel design artifact, called OntoMaven,[5] as a relevant and original contribution in distributed ontology engineering. The approach is grounded in the rigorous methods of Maven's engineering approach, which has already proven its value and success in software engineering. The OntoMaven approach supports ontology engineering in the following ways:

- OntoMaven remote repositories enable distributed publication of ontologies as **ontology development artifacts** on the Web, including their metadata information about life cycle management, versioning, authorship, provenance, licensing, knowledge aspects, dependencies, etc.
- OntoMaven local repositories enable the reuse of existing ontology artifacts in the users' local ontology development projects.
- OntoMaven's support for the different development phases from the design, development to testing, deployment and maintenance provides a flexible life cycle management enabling iterative agile ontology development methods, such as COLM [6], with support for collaborative development by, e.g., OntoMaven's dependency management, version management, documentation and testing functionalities, etc.
- OntoMave plug-ins provide a flexible and light-weight way to extended the OntoMaven tool with existing functionalities and tools, such as semantic version management (e.g., SVont - Subversion for Ontologies [7, 8]), semantic documentation (e.g., SpecGen Concept Grouping [10]), dependency management of aspect-oriented ontology artifacts (e.g., [11]), automated testing (e.g., with the W3C OWL test cases and external reasoners such as Pellet), etc.
- Maven's API allows easy integration of OntoMaven into other ontology engineering tools and their integrated development environments (IDE).

The further paper is structured as follows: Sect. 2 describes related work. Section 3 describes the design of OntoMaven based on Maven's remote and local repositories, the Project Object Model (POM), plug-ins, and important functional concepts of the solution approach for OntoMaven ontology development. Section 4 proves the

[4]http://maven.apache.org/.

[5]http://www.corporate-semantic-web.de/ontomaven.html.

feasibility of the proposed concepts with a proof-of-concept implementation of the OntoMaven design artifact. Section 5 compares the OntoMaven functionalities to the tool support of the major existing ontology engineering tools, namely Protege,[6] Swoop,[7] and Top Braid Composer.[8] This highlights the strengths of OntoMaven with the open approach to model, manage, and reuse ontology (modules) as **ontology development artifacts** including their metadata descriptions in the POM, dependency management, versioning, documentation, etc. Finally, Sect. 6 summarizes the current OntoMaven work and discusses future research.

2 Related Work

There are many existing ontology engineering methodologies and ontology editors available. With its Maven-based approach for structuring the development phases into different goals providing different functionalities during the development project's life cycle, OntoMaven supports in particular agile ontology development methods, such as RapidOWL [12] and COLM [6], as well as development methods which are inherently based on modularization such as aspect-oriented ontology development [11] (see also chapter "Aspect-Oriented Ontology Development").

RapidOWL provides a light-weight framework consisting of values (e.g., transparency and simplicity), principles (such as incremental, organic, uniform, observable, WYSIWYM, and rapid feedback), and best practices (e.g., short releases, joint ontology design, adherence to modeling standards, and testing in the form of consistency checking), which are inspired by agile software development. In contrast to other approaches, RapidOWL does not define an ontology lifecycle model. The principles defined by RapidOWL have been deliberately designed to be independent of any application context, with the aim to maximize flexibility.

The *Corporate Ontology Lifecycle Methodology (COLM)* defines an agile ontology lifecycle model with strictly defined cyclic phases. COLM interprets the development cycle as an ongoing evolution of the ontology. A dedicated initial requirement analysis phase is not defined. Instead, it is integrated into the development cycle.

COLM provides for the elicitation of requirements in the form of user feedback, which is gathered during the use phases of the ontology. In this sense, COLM defines two cycles, namely a development or engineering cycle with the phases selection/integration/development, validation, and evaluation, and a usage cycle, including the phases deployment, population, feedback tracking, and reporting. In contrast to RapidOWL, COLM directly incorporates the application context of the ontology.

According to [13] the most popular ontology editors supporting the Semantic Web ontology languages (RDF, RDFS, OWL, SWRL) are Protege, Swoop and

[6]http://protege.stanford.edu/.

[7]http://www.mindswap.org/2004/SWOOP/.

[8]http://www.topquadrant.com/products/TB_Composer.html.

Top Braid Composer. Other editors support, e.g., visual ontology modeling such as Thematix Visual Ontology Modeler (VOM),[9] which enables UML-based ontology modeling based on the OMG Ontology Definition Metamodel (OMG ODM[10]), or lightweight domain specific vocabulary development, such as Leone[11] [8]. While the focus of OntoMaven is on supporting the backend-functionalities in ontology development projects, the focus of these editors is on the support of ontology modeling/representation with a user interface. Non of them is based directly on the Apache Maven concepts and its methods. While all of the editors also provide support for ontology repositories and reuse of existing ontologies by imports, OntoMaven mainly differs in the approach how it manages and declaratively describes the development phases, goals, and artifacts in a Maven Project Object Model (POM). Further implementation-specific and plug-in-specific differences are in the underlying details of the provided functionalities of OntoMaven such as POM-based dependency management, semantic versioning, semantic documentation etc. For a comparison see the evaluation Sect. 5.

The W3C Wiki lists several existing ontology repositories.[12] Further ontology repositories are, e.g., COLORE[13] for ontologies written in the ISO Common Logic (CL) ontology languages and Ontohub[14] which maintains a set of heterogenous ontologies. The current focus of these projects is on collection and listing of existing ontologies. Apart from simple search functionalities there is no support for repository-based ontology development which is the focus of OntoMaven and OntoMaven repositories.

New standardization efforts such as OMG Application Programming Interfaces for Knowledge Bases (OMG API4KB)[15] and OMG OntoIOP aim at the accessibility and interoperability of heterogenous ontologies via standardized interfaces and semantic transformations defined on the meta-level of the ontology models, e.g., by the Distributed Ontology Language (DOL) [14]. These approaches do not address ontology engineering directly, but can provide a standardized repository back-end for OntoMaven ontology development projects.

3 OntoMaven's Design and Concept

This section describes the approach and the concepts of the new design artifact, called OntoMaven, which adapts Apache Maven for the Maven-based ontology development and management of distributed OntoMaven ontology repositories. Maven is

[9]http://thematix.com/tools/vom/.

[10]http://www.omg.org/spec/ODM/.

[11]Leone - http://www.corporate-semantic-web.de/leone.html.

[12]http://www.w3.org/wiki/Ontology_repositories.

[13]http://stl.mie.utoronto.ca/colore/.

[14]http://ontohub.org/.

[15]www.omgwiki.org/API4KB/.

not just an automated build tool but also supports software artifact management and quality assurance in software projects. The main required functionalities provided by Maven are:

- source code compilation
- dependency management
- testing with test suites
- automated documentation and reporting
- installation and deployment of generated code.

The main design concepts of Maven are:

- The Project Object Model (POM) is the main declarative XML description for managing a project and its development artifacts. Based on the instructions in a POM file Maven automates the different project goals in the life cycle of a software development project.
- Maven plug-ins implement the functionality of the different Maven goals and lead to a modular and extensible architecture of Maven. The plug-ins are executed by Maven using the descriptions in the POM file. Maven has three predefined life cycles, namely the *Clean* life cycle, which cleans the project, the *Default* life cycle, which processes, builds, tests and installs locally or deploys remotely, and the *Site* life cycle, which reports, documents and deploys the created HTML documentation, e.g., on a Web server.
- Maven local and remote repositories manage the used plug-ins and artifacts including support for versioning and dependency management. The general approach is that libraries of existing software frameworks and Maven plug-ins which are required in a software development project are downloaded from distributed remote repositories to the local Maven repository so that Maven can work with these artifacts locally during the development. This distributed approach supports sharing and reuse of existing software artifacts. The information about the used artifacts and their remote addresses (typically a URL) as well as dependency information are described in the POM file of a project. The downloaded artifacts have their own POM files in order to support e.g., transitive dependencies.

In the following subsections we adapt the main concepts of Maven, so that they can be used in ontology development and ontology life cycle management. In particular, we focus on the (distributed) management of knowledge artifacts (ontologies/ontology modules) and their versioning, import and dependency management, documentation, and testing.

3.1 Management and Versioning of Ontology Artifacts

One of the design patterns in ontology engineering is the reuse of existing ontologies and ontology modules. Finding ontologies on the Web is supported, e.g., by specialized ontology search engines such as Swoogle[16] and Watson.[17] Since such found ontologies typically cannot be used directly, but need to modularized, refactored and enhanced, before they can be reused in an ontology development project, there is need for versioning and life cycle management of such ontologies. Furthermore, combinations with other existing ontologies (by ontology matchmaking and alignment) might lead to transitive dependencies which need to be described and managed. OntoMaven therefore adopts Maven's artifact concept. It describes and manages ontologies as ontology artifacts in a Maven Project Object Model (POM). The typical steps to add an ontology (module) as an OntoMaven artifact to a POM are:

1. Find ontology module(s)
2. Select the right module and version
3. Analyse and resolve dependencies of the modules
4. Declaratively describe the ontology artifact in a POM

Many ontology languages support imports or integration of distributed ontologies. For instance, the W3C Web Ontology Language (OWL) therefore has a specialized `owl:import` statement and the ISO Common Logic standard supports modularized imports by a segregation semantics which distinguishes the universe of discourse of the main ontology from the segregated universe of discourse of the imported ontology.

Typical recurring tasks which are automated by OntoMaven are in such modular import and reuse scenarios are,

- check the existence of the imported ontology (module) referenced by the defined URI in the import statement (and find alternative URLs from pre-configured repositories if the ontology does is not found at the import URI).
- management of ontologies/ontology modules as ontology artifacts in Maven repositories including their metadata descriptions such as versioning information.
- download of ontology artifacts from remote repositories (including transitive imports) to a local development repository in order to support offline development of ontologies.

Another important aspect in the agile and collaborative development of ontologies is the support for version management. Typical requirements are maintaining consistency and integrity, as well as provenance data management (e.g., authorship information) throughout the version history. Compared to version management solutions in software engineering which just maintain syntactic versions on the level of code line differences, the difficulty in knowledge engineering is that the versions and their differences need to be managed on a semantic level. Different syntactic versions of an ontology knowledge model might still have the same semantic interpretation. That is, a semantic version management system needs to compute the semantic difference, e.g., to detect and resolve version conflicts. The approach in OntoMaven is

[16]http://swoogle.umbc.edu/.

[17]http://kmi-web05.open.ac.uk/WatsonWUI/.

based on the ontology versioning tool *SVont*,[18] which is an extensions of the version management tool Subversion. [7, 8].

3.2 Import and Dependency Management

OntoMaven adopts the dependency management of Maven by describing the dependencies in an ontology development project from existing ontology artifacts in a POM. This is illustrated in the following listing from a POM example:

```
<project>
  ...
  <dependencies>
    <dependency>
        <groupId>de.onto.maven</groupId>
        <artifactId>TimeOntologie</artifactId>
        <version>1.0</version>
    </dependency>
...
</dependencies>
</project
```

The listing describes a dependency of an ontology artifact identified by the ID `TimeOntologie` version `1.0` which belongs to the group `de.onto.maven`.

It is possible to define multiple repositories in which OntoMaven will look for dependent ontology artifacts. If the defined ontology artifact is not found in the repository, users will be informed. They can start a search with the ontology name and URI defined in the artifact description in configured ontology search engines. OntoMaven supports Swoogle and Watson. The found ontologies can be added as new ontology artifacts to an OntoMaven repository.

3.3 Documentation

Despite first automated ontology matching and alignment approaches developing high quality ontologies still remains a manual knowledge modeling effort, where domain experts work together with knowledge engineers. Documentation is an important step which facilitates this work and in particular makes maintenance and reuse easier. The typical distinction is into *user documentation* and *technical documentation*. While the former supports the users of an ontology, e.g., in their task to

[18] http://www.corporate-semantic-web.de/svont.html.

populate the ontology with instance data, the latter, technical documentation, supports the ontology developer.

There exist several ontology documentation tools such as OWLDoc,[19] VocDoc,[20] or SpecGen,[21] which document ontologies on a technical level (much like tools such as JavaDoc in Java programming). Unfortunately, the lack of good documentation in the published ontologies on the Web makes reuse difficult, because the analysis and decision process on the applicability of a candidate ontology becomes very time-consuming. Therefore, additional automated support needs to be provided, e.g., for analysing larger ontologies on an abstract level by creating concept groupings which reduce the complexity of the ontology model. This groupings and summarizations of concepts provides the reader with an easier way to understand the ontology vocabulary. Typically such concept groups are additionally presented in easy to understand visualization formats, e.g., by tools such as OWLViz,[22] OntoGraf,[23] Sonivis OWL plugin,[24] SOVA,[25] TGVizTab,[26] or as UML models, e.g., by VOM.[27] Such concept grouping and visualisations can be used in the user documentation of an ontology. [10].

Maven supports the documentation phase and provides goals for creating and publishing automated reports. In OntoMaven we make use of SpecGen and the SpecGen extension[28] for automated concept grouping, in order to create the technical and user documentation in an OntoMaven plugin which is executed by the `mvn site` command.

3.4 Testing

Testing is an important phase in the ontology life cycle. In particular, in agile iterative development processes testing allows detecting inconsistencies, anomalies, improper design, as well as validation against, e.g., the intended results of domain experts' competency questions which are represented as ontology test cases. Maven supports a testing phase in which automated tests are executed and the results are reported by the Maven command `mvn test`.

[19]http://docpp.sourceforge.net/.
[20]http://kantenwerk.org/vocdoc/.
[21]https://github.com/specgen/specgen.
[22]http://www.co-ode.org/downloads/owlviz/.
[23]http://protegewiki.stanford.edu/wiki/OntoGraf.
[24]http://www.corporate-semantic-web.de/ontology-modularization-framework.html.
[25]http://protegewiki.stanford.edu/wiki/SOVA.
[26]http://users.ecs.soton.ac.uk/ha/TGVizTab/.
[27]http://thematix.com/tools/vom/.
[28]http://www.corporate-semantic-web.de/concept-grouping.html.

The W3C OWL recommendation[29] defines a collection of test cases with different test types and tests. As standard test types OntoMaven by default supports the W3C OWL test cases *syntax checker*, *consistency checker*, and *entailment test*. The produced test results are compliant to the W3C recommendation and the created test reports show if the ontology model is `consistent`, `inconsistent` or if the result is `unknown`. Further test types can be implemented as Maven plug-ins and added to the OntoMaven projects test suites by the user.

4 Proof-of-Concept Implementation - OntoMaven PlugIns

The implementation of OntoMaven extends and adapts Maven [15], so that it supports the management of ontology modules in Maven repositories. This section describes how the OntoMaven approach and the above described concepts are implemented using *Maven repositories* and the *Maven plug-in* extension mechanism. A Maven plug-in is a collection of one or more goals. For instance, the plug-in `archetype` implements the goals `create` and `generate` which create a Maven project. The implementation of a Maven plug-in is done in an *Maven Plain Old Java Object (MOJO)*. Maven supports the automated generation of a Mojo project with the goal `generate` in the Maven plugin `Archetype`:

```
Mvn archetype:generate -DinteractiveMode=false
-DarchetypeArtifactId=maven-archetype-mojo -DgroupId=[] -DartifactId=[]
```

By defining the `groupId` and the `artifactID` the command creates a Mojo project with a Mojo class which is used for the plug-in implementation.
In the OntoMaven approach, the phases and goals, which the plug-in implements, are defined by JavaDoc annotations in the source code of the Mojo class. For instance, the following annotations define that the implemented plug-in is used in the phase `test` and that is has a goal called `test-syntax`:

```
@phase test //plug-in used in test phase
@goal test-syntax // goal with the name "test-syntax"
```

Parameters are used to configure the plug-in execution. For instance, the following code snippet defines a required parameter `compliancemode` with the default value `strict`:

```
*@parameter expression = compliancemode
*default-value="strict"
*@required
```

[29] http://www.w3.org/TR/owl-test/.

Such plug-in parameters can be configured in a POM.xml file or directly when calling a goal, e.g., `mvn ... -Dcompliancemode=strict`. An implemented plug-in can be installed using Maven `mvn install` and the plug-in goals can be integrated into the POM.xml of an OntoMaven project, as the following example listing shows for the plug-in SVontPlugin and the goal semantic-diff:

```
<build>
    <plugins>
        <plugin>
            <groupId>de.csw.ontomaven</groupId>
            <artifactId>SVontPlugin</artifactId>
            <version>1.0-SNAPSHOT </version>
            <executions>
              <execution>
               <goals>
                <goal>semantic-diff</goal>
               </goals>
              </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

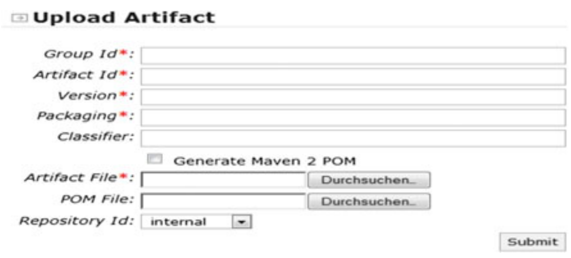
The following subsections provide further details about the proof-of-concept implementations of the main plug-ins in OntoMaven. We first describe the OntoMaven repositories which are the persistence and back-end layer for storing and managing ontologies.

4.1 OntoMaven Repositories

OntoMaven can use all Maven compliant repositories. One of the strengths of Maven is that is uses a folder structure following a standard folder layout for its repositories; sources are in `$basedir/src/main/java`, resources in `$basedir/src/main/resource`, tests in `$basedir/src/test`, classes in `$basedir/target/classes`, and packaged libraries in `$basedir/target/`.

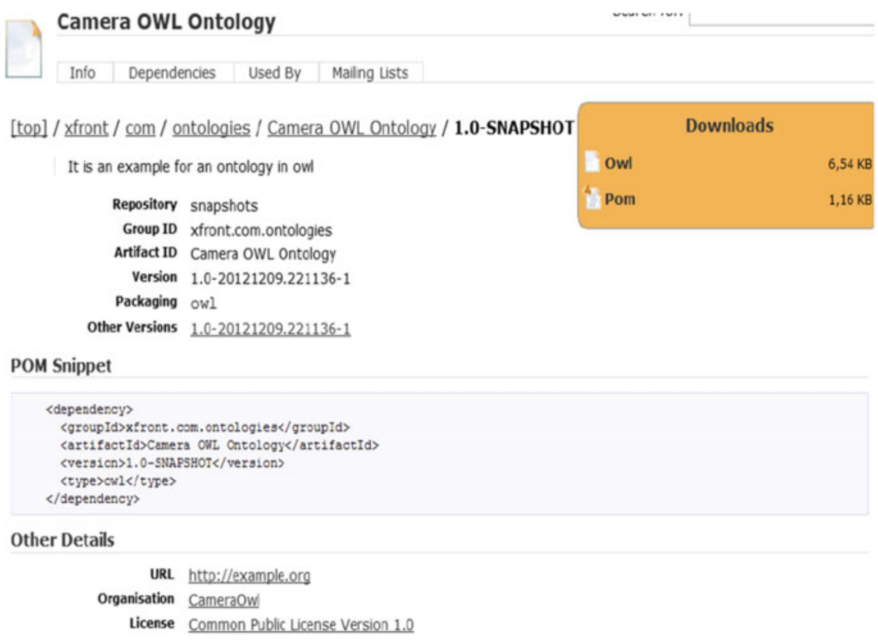
For the OntoMaven proof-of-concept implementation we adapted the Apache Archiva Build Artifact Repository Manager[30] as a managing tool providing a user interface for the OntoMaven repositories. It supports finding and managing OntoMaven artifacts. Figure 1 shows the upload user interface.

[30] http://archiva.apache.org/.



Group Id*:	
Artifact Id*:	
Version*:	
Packaging*:	
Classifier:	
	Generate Maven 2 POM
Artifact File*:	
POM File:	
Repository Id:	internal

Fig. 1 Archiva user interface - ontology artifact upload



Camera OWL Ontology

[top] / xfront / com / ontologies / Camera OWL Ontology / 1.0-SNAPSHOT

It is an example for an ontology in owl

Repository	snapshots
Group ID	xfront.com.ontologies
Artifact ID	Camera OWL Ontology
Version	1.0-20121209.221136-1
Packaging	owl
Other Versions	1.0-20121209.221136-1

Downloads

Owl	6,54 KB
Pom	1,16 KB

POM Snippet

```
<dependency>
  <groupId>xfront.com.ontologies</groupId>
  <artifactId>Camera OWL Ontology</artifactId>
  <version>1.0-SNAPSHOT</version>
  <type>owl</type>
</dependency>
```

Other Details

URL	http://example.org
Organisation	CameraOwl
License	Common Public License Version 1.0

Fig. 2 Archiva user interface - management of ontology artifact

Via this form an ontology can be uploaded to an OntoMaven repository together with its POM file. The artifact's metadata contains information about the group id, artifact id, version, packaging and optional additional classifier information. The POM provides all necessary information about the artifact and its dependencies. In OntoMaven these dependencies are used to describe (transitive) imports from an ontology, which are resolved by the OntoMvnImport plug-in (see Sect. 4.2) and are defined in the ontology's POM file.

Figure 2 gives an example of the Archiva user interface showing the management information of an ontology artifact called `Camera OWL Ontology`. Under the interface menu link `Dependencies` the dependencies of this ontology can be found.

Once managed in an online OntoMaven repository, an ontology artifact can be used in any OntoMaven ontology development project. The following listing gives an example how a remote repository can be configured and a dependency to an ontology artifact (here the Camera-OWL-Ontology) can be defined in the POM.xml document of a project.

```
<profiles>
    <profile>
        <id>2</id>
        <activation>
            <activeByDefault>true</activeByDefault>
        </activation>
        <repositories>
            <repository>
                <snapshots>
                    <enabled>true</enabled>
                </snapshots>
                <id>snapshots</id>
                <name>OntoMaven Snapshot Repository</name>
                <url>www.corporate-semantic-web.de/repository/snapshots/</url>
            </repository>
        </repositories>
    </profile>
</profiles>

<dependencies>
    <dependency>
        <groupId>xfront.com.owl.ontologies</groupId>
        <artifactId>Camera-OWL-Ontology</artifactId>
        <version>1.0-SNAPSHOT</version>
        <type>owl</type>
    </dependency>
</dependencies>
```

4.2 OntoMvnImport

This plug-in implements the imports of ontologies into the Maven repositories. It is also checks if the import statements in the ontology including transitive imports can be resolved. Therefore, it maintains an updated list of reference URIs to the ontology resources loaded to the repository. This list follows the OASIS XMLCatalog standard which also specifies a technique for the automated replacement of external references in XML documents. An XML parser validates if defined replacement rules in an XMLCatalog apply to the references in the validated XML document and in case they apply, it replaces these references with the references defined in the XMLCatalog. In OntoMaven we use this automated replacement technology to replace the URI references to imported **external ontologies** with references to the **internal ontology artifacts**, which are locally managed in an OntoMaven repository after they have been loaded by the plug-in. This replacement approach avoids the continuous import and use of external ontologies during an OntoMaven development project. As an example the following catalog entry defines that the original reference to the imported ontology `example.owl` can be replaced by the URI referencing the stored ontology artifact in the local repository.

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog">
    <system systemId="www.example.com/example.owl" uri="src/resource/owl"/>
</catalog>
```

After the first loading of an ontology as repository artifact, including all transitive imports which are resolved and stored as dependent ontology artifacts, by the OntoMvnImport plug-in, the plug-in always checks if there is an ontology artifact listed in the XMLCatalog. If there is an existing reference to an ontology artifact, it will use it instead of any externally referenced ontology. A special situation is, if the import statement cannot be resolved, e.g., because the ontology is no longer existent under the given reference. In this case the plug-in notifies the user.

The following listing shows how the plug-in can be used in a OntoMaven POM. In the `configuration` it defines the input ontology and sets the `local` parameter to true, indicating that the ontology should be loaded to the local repository and that the local version of the ontology should be used.

```
<build> <plugins> <plugin>
    <groupId>de.csw.ontomaven</groupId>
    <artifactId>OntoMvnImport</artifactId>
    <version>1.0-SNAPSHOT</version>
    <configuration>
        <owlfile>src/resource/reputation.owl</owlfile>
        <local>true</local>
    </configuration>
    <executions>
        <execution>
            <goals>
                <goal>owlimport</goal>
            </goals>
        </execution>
    </executions>
</plugin> </plugins> </build>
```

4.3 OntoMvnSvn

The `OntoMvnSvn` plug-in provides ontology versioning support for OntoMaven. As discussed in Sect. 3.1 standard (code) versioning tools such as Concurrent Version System (CVS) and Subversion cannot be directly used, because version differences are only computed syntactically, but not semantically, as it is required for versioning interpreted knowledge models such as ontologies.

We therefore implemented an extension to Subversion called SVont[31] [7] which can compute semantic differences[32] and which can version ontologies. SVont supports typical Subversion commands such as `checkout`, `status`, `diff`, `commit`, and `info`. In the following the implementation of the commands `status` and `diff` are described in more detail.

For the *status* command the OntoMvnSvn plug-in first does a repository checkout of the ontology to a temporal folder and then compares the repository version with the currently developed working version. The possible status results are *identical* and *changed*. This status information is used by the plug-in to either report *ontology changed* or *ontology is up-to-date*.

The *diff* command first does a checkout of the committed repository version. It then computes the semantic difference to the working version. Therefore, the OntoMaven implementation is using the semantic difference computation implemented by Svont. The OntoMvnSvn plug-in goal `diff` additionally performs a dependency analysis. It lists all dependencies, e.g., dependencies by domain and range properties and subclass relations as the following example shows:

```
-------------------------- DIFF INFORMATION --------------------
Ontology File : ...\...\...\...\...\camera.owl
================== ACTUAL CHANGES =========================
Axioms were added to the repository, or deleted from the working
copy.
SubClassOf(<www.xfront.com/owl/ontologies/camera/#Money>
owl:Thing)
Declaration(Class(<www.xfront.com/owl/ontologies/camera/#Money>))
=======================================================================
--------- MORE INFO --------------------------------
The above changes of the OWL classes are dependent on the following
axiom.
currency <------ DataProperty (Domain)
cost <------ ObjectProperty (Range)
--------------------------------------------------------------------
```

4.4 OntoMvnReport

The plug-in is implemented as Maven report plug-in.[33] The goal `site` of this plug-in creates four different documentations about the ontology - a general *project documentation*, an *ontology report summary*, a *technical report*, and an *ontology visualization*.

[31] http://www.corporate-semantic-web.de/svont.html.

[32] For the description logic $\mathcal{EL}$.

[33] http://docs.codehaus.org/display/MAVENUSER/Write+your+own+report+plugin.

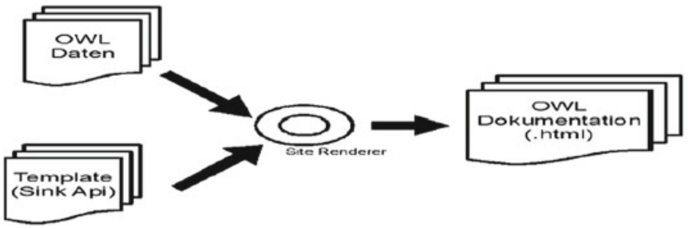
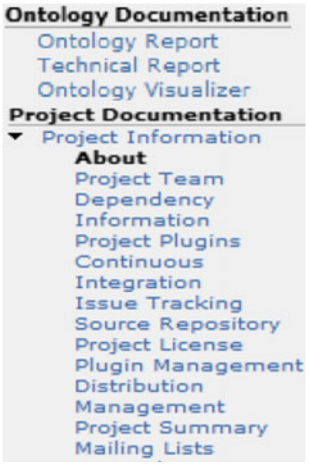


Fig. 3 Simplified diagram for the report rendering in the OntoMvnReport Plug-In

Fig. 4 OntoMaven project and ontology documentation menu



For rendering the ontology information into an HTML documentation it uses the Sink API[34] as illustrated in Fig. 3.

With the Maven Site Descriptor[35] the layout and content menus in OntoMaven reports can be adapted.

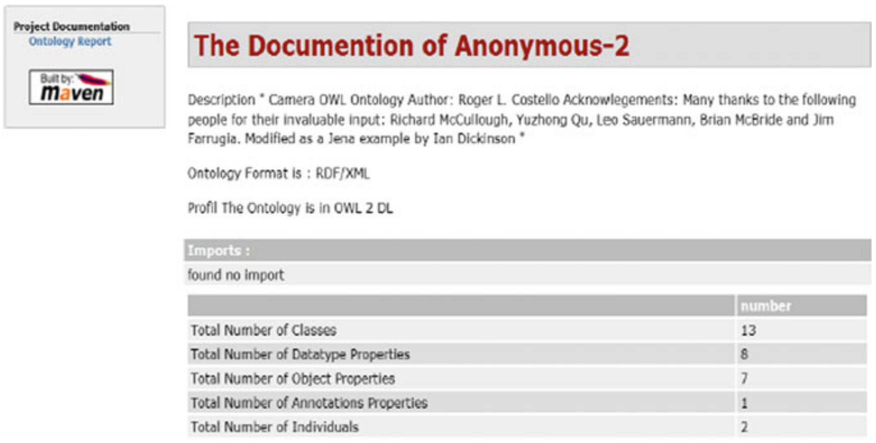
The general *ontology project documentation* is created from the description of the ontology artifact in the POM file, which includes project metadata about, e.g., the project, project team, dependencies, the plug-ins, issues, source repositories, licenses, ontology developers and their roles, etc. The following listing gives an example of typical project's metadata in a POM which is used in the project documentation which can be selected from the report menu (see Fig. 4):

[34]http://maven.apache.org/doxia/doxia/doxia-sink-api/.

[35]http://maven.apache.org/plugins/maven-site-plugin/examples/sitedescriptor.html.

```
<description>here's the descripton of an ontology </description>
<organization>
    <name>Corporate Semantic Web, Freie Universitt Berlin</name>
    <url>www.corporate-semantic-web.de</url>
</organization>
<inceptionYear>2013</inceptionYear>
<licenses>
    <license>
        <name>LGPL-3.0</name>
        <url>www.gnu.org/licenses/lgpl.txt</url>
    </license>
</licenses>
<developers>
    <developer>
        <name>Adrian Paschke</name>
        <email>paschke@inf.fu-berlin.d</email>
        <organization>Corporate Semantic Web</organization>
        <organizationUrl>www.corporate-semantic-web.de/
        </organizationUrl>
        <roles>
            <role>developer</role>
        </roles>
    </developer>
</developers>
```

The *ontology report summary* is created by the goal `ontologyreport`. Figure 5 shows an example ontology summarization which gives an overview about the general description, the format, the semantic profile, imported ontologies and a summary about the ontology's statistics (number of classes, datatype properties, object properties, etc.).



The Documention of Anonymous-2

Description " Camera OWL Ontology Author: Roger L. Costello Acknowlegements: Many thanks to the following people for their invaluable input: Richard McCullough, Yuzhong Qu, Leo Sauermann, Brian McBride and Jim Farrugia. Modified as a Jena example by Ian Dickinson "

Ontology Format is : RDF/XML

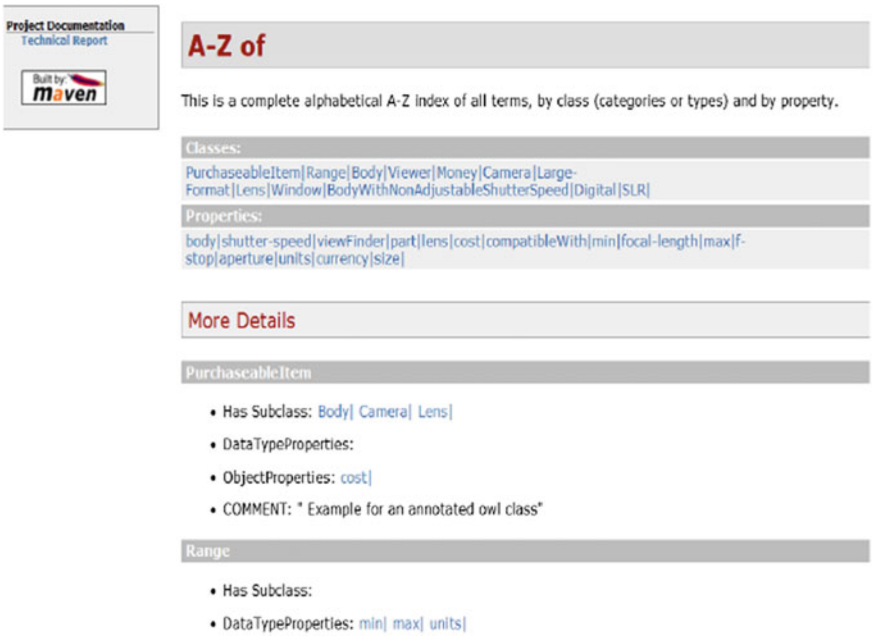
Profil The Ontology is in OWL 2 DL

Imports :

found no import

	number
Total Number of Classes	13
Total Number of Datatype Properties	8
Total Number of Object Properties	7
Total Number of Annotations Properties	1
Total Number of Individuals	2

Fig. 5 OntoMaven ontology report summary



A-Z of

This is a complete alphabetical A-Z index of all terms, by class (categories or types) and by property.

Classes:
PurchaseableItem|Range|Body|Viewer|Money|Camera|Large-Format|Lens|Window|BodyWithNonAdjustableShutterSpeed|Digital|SLR|

Properties:
body|shutter-speed|viewFinder|part|lens|cost|compatibleWith|min|focal-length|max|f-stop|aperture|units|currency|size|

More Details

PurchaseableItem

- Has Subclass: Body| Camera| Lens|
- DataTypeProperties:
- ObjectProperties: cost|
- COMMENT: " Example for an annotated owl class"

Range

- Has Subclass:
- DataTypeProperties: min| max| units|

Fig. 6 OntoMaven technical report

For the documentation of the ontology, the plug-in uses existing automated ontology documentation tools. We have integrated the SpecGen ontology documentation tool which creates a HTML page containing detailed information about the classes and the properties. We further extended SpecGen with various algorithms for creating structure based concept groupings. [10, 16] These groupings are used as basis for a visual documentation of the ontology. To support this process of creating such concept groups for the documentation of ontologies we extended the SpecGen tool with an automatic concept grouping functionality[36] and embedded it for the OntoMaven documentation.

A more detailed insight is given by the *technical ontology report*, which is created by the goal `technicalreport`. This goal produces a listing of classes and properties as shown in Fig. 6. By clicking on a particular class or property the technical details about it are shown.

The goal `visualizer` produces a network visualization of the ontology concepts (classes) and its relations using different graph visualizations.[37] Figure 7 gives a visualization example.

The following listing shows how to use the `OntoMvnReport` plugin in a project POM.xml

[36] http://www.corporate-semantic-web.de/concept-grouping.html.

[37] http://www.corporate-semantic-web.de/ontology-modularization-framework.html.

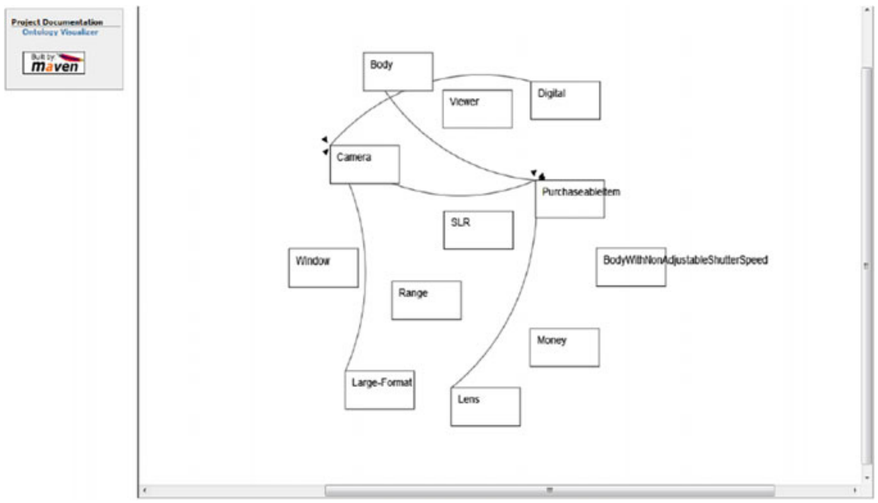


Fig. 7 OntoMaven ontology visual report

```
<reporting><plugins><plugin>
<groupId>de.nbi.MvnOnt</groupId>
<artifactId>MvnOwlReport</artifactId>
<version>1.0-SNAPSHOT</version>
<reportSets>
    <reportSet>
        <configuration></configuration>
        <reports>
            <report>ontologyreport</report>
            <report>technicalreport</report>
            <report>visualizer</report>
        </reports>
    </reportSet>
</reportSets></plugin></plugins></reporting>
```

The produced reports can be found in the Maven folder `target/site`.

4.5 *OntoMvnTest*

The *OntoMvnTest* plug-in implements functionalities for the test phase. The plug-in executes the configured test using the goal `test`. It is also used internally in other phases such as the `package` goal. The plug-in implementation uses the Pellet reasoner[38] to execute the ontology test cases.

[38] http://clarkparsia.com/pellet/.

As default test suites the plug-in supports the W3C OWL Test Cases.[39] This test collection contain different types of test cases, such as a test that determines and returns the OWL sublanguage, tests for inconsistency checks, and entailment tests, which test if the intended conclusions (represented by an output ontology) are entailed in the input ontology model. For instance, the intended entailment test result values are `Entailment` (positive test result) or `NoEntailment` (negative test result). The following listing shows how the plug-in can be used in a POM.xml.

```
<build>
<plugins>
    <plugin>
        <groupId>de.csw.MvnOnt</groupId>
        <artifactId>MvnOwlTest</artifactId>
        <version>1.0-SNAPSHOT</version>
        <configuration>
            <owlfile>owl/1a.owl</owlfile>
        </configuration>
        <executions>
            <execution>
                <goals>
                    <goal>owltest</goal>
                </goals>
            </execution>
        </executions>
    </plugin>
    <plugin>
        <groupId>de.csw.MvnOnt</groupId>
        <artifactId>MvnOwlEntailment</artifactId>
        <version>1.0-SNAPSHOT</version>
        <configuration>
            <premise_file>owl/1a.owl</premise_file>
            <conclusion_file>owl/1aconclusion.
            owl</conclusion_file>
        </configuration>
        <executions>
            <execution>
                <goals>
                    <goal>owlentailment</goal>
                </goals>
        </execution>
    </executions>
</plugin></plugins></build>
```

[39] http://www.w3.org/TR/owl-test/.

Table 1 Functional comparison of OntoMaven with ontology development tools

	OntoMaven	Protege	Swoop	Top Braid Composer
Repositories	Yes (local and remote)	Yes (local and remote)	No	Yes (by Allegro Graph 4 PlugIn)
Reuse (Import)	Yes (dependency management)	Yes	Yes	Yes
Collaboration Support (Versioning)	Yes (semantic diff)	No	No	No
Documentation	Yes (text and visual)	Yes (text and visual)	Yes (only text)	Yes (text and visual in Maestro version)
Testing	Yes	Yes	Yes	Yes
Extensibility	Yes	Yes (many existing plugins)	Yes	Yes (commercial)
	WebOnto	OilEd	OntoSaurus	WebODE
Repositories	No (central server)	No	No	No
Reuse (Import)	No	Yes	No	Yes
Collaboration Support (Versioning)	Yes (web-based, support for discussion-based collaboration)	No	No	Yes
Documentation	No	Yes (HTML export)	No	Yes
Testing	No	No	Yes (simple consistency maintenance)	No
Extensibility	No (server, Java applet)	No	No	Yes (extensible workbench)
	OntoEdit	Neon Toolkit		
Repositories	No (but database adaptors)	No		
Reuse (Import)	Yes	Yes		
Collaboration Support (Versioning)	Yes	Yes		
Documentation	No	Yes		
Testing	No	Yes		
Extensibility	Yes (plugins)	Yes (plugins)		

5 Evaluation

OntoMaven is not a full ontology development tool as e.g., Protege, Swoop and Top Braid Composer, which provide a development user interface. Instead OntoMaven's has its strength in the management of distributed ontology modules including support for reuse (transitive imports), dependency management and collaboration (semantic versioning).

Table 1 compares OntoMaven to the most used ontology development tools by its functional support in typical ontology engineering life cycles.

6 Conclusion

Apache Maven is a widespread and highly successful tool in Software Engineering for build automation and development project life cycle management. This paper has adapted the Maven approach and concepts for Knowledge Engineering in (agile) ontology development. The contribution is a new design artifact called OntoMaven which has been implemented as a proof-of-concept implementation.

OntoMaven uses a Project Object Model (POM) XML file to describe the ontology project being developed, its dependencies on other external ontology modules, the development life cycle order, directories, and required plug-ins. It comes with pre-defined targets for performing certain well-defined tasks in typical agile ontology development phases.

OntoMaven dynamically downloads distributed ontologies and Maven plug-ins from one or more OntoMaven remote repositories and stores them in a local cache, the local repository. This local cache of downloaded artifacts can be further updated with ontology artifacts created by local projects as well as developed artifacts can be uploaded to remote OntoMaven repositories, so that they can be shared and reused.

OntoMaven is built using Maven's plugin-based architecture that allows it to make use of any application controllable through standard input. The proof-of-concept of OntoMaven implements several useful plugins which interface with existing ontology development tools and functionalities such as the plug-ins *OntoMvnImport*, *OntoMvnSVN*, *OntoMvnReport*, and *OntoMvnTest*.

In future research we plan to study the benefits of the OntoMaven support for ontology engineers in real agile ontology development projects and compare it with other development approaches. We plan to quantify the results on the basis of ontology development costs (in person months) and compare them against the estimated cost models (based on our work on Agile ONTOCOM 2[40] [17]).

In our aspect-oriented ontology development research [11] we plan to make use of OntoMaven for the distributed management of ontology modules and their description as *aspect-oriented ontology artifacts* in the POM.

[40]http://www.corporate-semantic-web.de/corporate-ontology-engineering.html.

Acknowledgements This work has been partially supported by the InnoProfile project "Corporate Semantic Web" and "Corporate Smart Content" funded by the German Federal Ministry of Education and Research (BMBF).

References

1. Weigand, H., Paschke, A.: The pragmatic web: putting rules in context. In: RuleML, pp. 182–192 (2012)
2. Paschke, A., Boley, H.: Rule responder: rule-based agents for the semantic-pragmatic web. Int. J. Artif. Intell. Tools **20**(6), 1043–1081 (2011)
3. Paschke, A., Boley, H., Kozlenkov, A., Craig, B.L.: Rule responder: ruleml-based agents for distributed collaboration on the pragmatic web. In: ICPW, pp. 17–28 (2007)
4. De Nicola, A., Missikoff, M., Navigli, R.: A software engineering approach to ontology building. Inf. Syst. **34**(2), 258–275 (2009). doi:10.1016/j.is.2008.07.002
5. Paschke, A., Coskun, G., Heese, R., Luczak-Rsch, M., Oldakowski, R., Schfermeier, R., Streibel, O.: Corporate semantic web: towards the deployment of semantic technologies in enterprises. In: Du, W., Ensan, F. (eds.) Canadian Semantic Web, pp. 105–131. Springer, US (2010). doi:10.1007/978-1-4419-7335-1_5
6. Luczak-Rsch, M., Heese, R.: Managing ontology lifecycles in corporate settings. In: Pellegrini, T., Auer, S., Tochtermann, K., Schaffert, S. (eds.) Networked Knowledge - Networked Media. Studies in Computational Intelligence, vol. 221, pp. 235–248. Springer, Berlin (2009). doi:10.1007/978-3-642-02184-8_16
7. Luczak-Rsch, M., Coskun, G., Paschke, A., Rothe, M., Tolksdorf, R.: Svont - version control of owl ontologies on the concept level. In: Fhnrich, K.P., Franczyk, B. (eds.) GI Jahrestagung (2), LNI, vol. 176, pp. 79–84. GI (2010). http://dblp.uni-trier.de/db/conf/gi/gi2010-2.html#Luczak-RoschCPRT10
8. Paschke, A., Coskun, G., Hartrampf, D., Heese, R., Luczak-Rösch, M., Rothe, M., Oldakowski, R., Schäfermeier, R., Streibel, O.: Realizing the corporate semantic web: prototypical implementations **TR-B-10-05**, 1–49 (2010). http://edocs.fu-berlin.de/docs/receive/FUDOCS_document_000000005563
9. Coskun, G., Luczak-Rösch, M., Heese, R., Paschke, A.: Applying ontology modularization for corporate ontology engineering. In: Proceedings of the Intl. Conference on Semantic Systems (I-SEMANTICS 2009), pp. 669–674. Graz, Austria (2009). http://www.i-semantics.at/2009/papers/applying_ontology_modularization.pdf
10. Coskun, G., Rothe, M., Paschke, A.: Ontology content "at a glance". In: Donnelly, M., Guizzardi, G. (eds.) Proceedings of the 7th International Conference on Formal Ontology in Information Systems, pp. 147–159. IOS Press, Graz, Austria (2012). doi:10.3233/978-1-61499-084-0-147
11. Schäfermeier, R., Paschke, A.: Towards a unified approach to modular ontology development using the aspect-oriented paradigm. In: 7th International Workshop on Modular Ontologies (WoMO 2013) (2013)
12. Auer, S.: The rapidowl methodology–towards agile knowledge engineering. In: WETICE, pp. 352–357 (2006)
13. Khondoker, M.R., Mueller, P.: Comparing ontology development tools based on an online survey. In: Proceedings of the World Congress on Engineering 2010 (WCE 2010) (2010)
14. Lange, C., Kutz, O., Mossakowski, T., Grüninger, M.: The distributed ontology language (dol): ontology integration and interoperability applied to mathematical formalization. In: CoRR (2012). arXiv:abs/1204.5093
15. Kilic, O.: Erweiterung von maven zur toolbasierten verwaltung von ontologiemodulen (2013)
16. Coskun, G., Rothe, M., Teymourian, K., Paschke, A.: Applying community detection algorithms on ontologies for identifying concept groups. In: WoMO, pp. 12–24 (2011)

17. Paschke, A., Coskun, G., Marko, H., Heese, R., Oldakowski, R., Schäfermeier, R., Streibel, O., Teymourian, K., Todor, A.: Corporate semantic web report vi: Validation and evaluation **TR-B-13-01**, 1–64 (2013). http://edocs.fu-berlin.de/docs/receive/FUDOCS_document_000000018374

Non-distracting, Continuous Collection of Software Development Process Data

Andrea Janes

Abstract Knowledge management initiatives often fail when companies lack time and resources to focus on the meaning, implications, capturing and sharing of organizational knowledge management. This problem becomes even more severe when dealing with software development companies: software is invisible, which makes it difficult to reason and to communicate about it. It is hard to understand status, e.g., what the current state of the project is, which difficulties exist, and which problems might be in front of us. This is why we need measurement to obtain data about software, how it is created, and how it is used. This chapter presents non-distracting, automatic measurement, which is based on the extension of code editors or the instrumentation of source code of products, to log how developers or users are interacting with the software. We present two examples how data was collected, analyzed and interpreted. The here discussed methods describe our experiences in developing systems that support software development teams to collect and organize knowledge about their software development process based on non-disturbing, automatic data collection technologies, dashboards, and the Goal-Question-Metric approach.

1 Introduction

Fred Brooks states in his seminal paper "No Silver Bullet—Essence and Accident in Software Engineering" [1] that "building software will always be hard". Two reasons that Brooks mentions are the complexity and invisibility of software. According to Brooks, complexity and invisibility are essential properties, not accidental ones. They are essential because they are inherent to the nature of the software and not just and accident, i.e., a difficulty that we have today in software production, but that we can overcome in the future with better tools.

A. Janes (✉)
Free University of Bozen-Bolzano, Universitätsplatz 1, 39100 Bozen, Italy
e-mail: ajanes@unibz.it

G.J. Nalepa and J. Baumeister (eds.), *Synergies Between Knowledge Engineering and Software Engineering*, Advances in Intelligent Systems and Computing 626, https://doi.org/10.1007/978-3-319-64161-4_13

Complexity and invisibility make it difficult to reason and to communicate about it. It is hard to understand status, e.g., what the current state of the project is, which difficulties exist, and which problems might be in front of us. It is also difficult to understand progress: the lack of a tangible product "means that it is very easy for the project to proceed for a considerable time before problems become apparent, and without it being possible to verify that the passing of time and expenditure of money correlate with progression of the project in the desired direction [2]."

Another consequence of the invisibility of software is that it can produce a perception "that anything and everything is possible" [2]: there seem to be no real constraints, which can lead to unrealistic expectations and over-ambitious projects.

To overcome the mentioned difficulties, it is important to measure software production and execution: measurement is the first step to describe and visualize what is invisible, to be able reason and communicate about complex software.

In this chapter, we adopt the terminology coined by Milan Zeleny, who distinguishes data, information, knowledge, and wisdom [3]. Each one of these terms represents a higher level of understanding [4] and is often depicted as a pyramid, as in Fig. 1.

The term "data" refers to symbols or signs. Examples of data are "9425277", "mountain", or "sensor 2". Data alone are of no use, therefore Zeleny calls data "know-nothing".

Information adds context to data so that they become useful. If we add context to the examples of data above and e.g., know that we want to bake bread, we understand that we are looking at a list of ingredients: water, flour, salt, spices, etc. This level of understanding is also called "know-what". It also includes the recipe for making bread that describes a number of steps to do with the ingredients.

Knowledge is described as "know-how", it refers "to observer's distinction of 'objects' (wholes, unities) through which he brings forth from the background of experience a coherent and self-consistent set of coordinated actions [3]." Knowledge connects data and information into a network of relations. In the bread baking example, knowledge describes the state that one knows several recipes, understands the functions that the different ingredients have in the recipe and understands what happens if he forgets to add one ingredient.

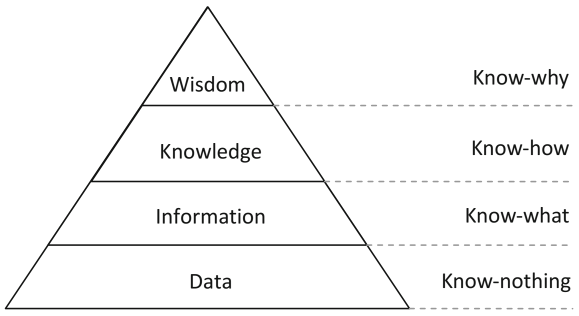


Fig. 1 The data, information, knowledge, wisdom pyramid [3]

Wisdom goes again one step further and links knowledge with a goal. In our taxonomy, wisdom is described as "know-why": we know why the different ingredients work and why we are using them. In our example this would mean that we know why we are using water, flour, salt, spices to make bread and not croissants.

According to Ackoff, data, information, and knowledge are past-oriented, while wisdom is future oriented [4].

The goal of this chapter is to present an non-distracting approach to continuously discover information, knowledge, and wisdom in software engineering processes.

The focus lies on "non-distracting", that means that we concentrate on an approach that does not involve *asking* programmers, users, project managers, etc. but to *measure* their interactions with various software systems to infer software engineering information, knowledge, and wisdom.

Asking about (or requiring to manually document) software development or software usage activities forces programmers or users to switch between the task they want to accomplish and the task of collecting data [5].

Task switching requires time and leads to a performance cost [6–10]. It creates costs because workers need time to "reorient" [11–13]. Stephen Jenkins [14] defines "deliberately planned, chronic interruptions" even as "worst-case scenario" and advices to "never let people work on more than one thing at once."

This means that there is a trade-off between asking often (highly distracting but obtaining data that the interviewed person still remembers precisely) or asking rarely (less distracting but risking that the reported cannot be remembered precisely anymore).

Using Zeleny's terminology, measurement helps to obtain data and information that is precise, but requires analysis to infer knowledge and wisdom. For example, instead of asking a programmer: "Why did you not test this function?" (know-why), we can collect data about all not tested functions (know-what) to infer that only certain components are not tested or that nobody writes tests before the end of release deadlines.

The rest of this chapter is structured as follows: Sect. 2 illustrates the main concepts used in this chapter as well as the technological choices we made to collect data. Section 3 illustrates examples how we use the collected data to discover information, knowledge, and wisdom within software engineering. Section 4 discusses advantages and disadvantages of the applied method, Sect. 5 concludes this chapter with final remarks.

2 Measurement

Using measurement we are able to collect the "know-what": for example, that a programmer wrote a new method or that a user clicked on a button. The goal is to infer "know-how" and "know-why", which we describe in the next section. This section describes the adopted approaches for to collect the necessary data.

The goal of this chapter is to present an non-distracting approach to continuously discover information, knowledge, and wisdom in software engineering processes.

Before we describe *how* data are collected, we need to define which data we need. Not having the right data at hand is costly: decisions have to be taken based on experience and gut feeling instead of facts. But the opposite, collecting all data, is also costly: one has to spend time developing tools to collect it, to store it, (hopefully) read, and use it for decision making. Therefore, it is important to collect only the data that are needed and with the needed precision.

One tool that can be used for this purpose is the Goal-Question-Metric approach [15]. In this approach, data collection is defined at three levels [5]:

1. **Conceptual** level (goal): defines what and why we study. What is studied is the "object of study", the specific products, processes, and resources. Why something is studied identifies the reason, the different aspect taken into consideration, the considered point of views, and the environment.
2. **Operational** level (question): here there are the questions that define (a) what parts of the object of study are relevant, and (b) what properties of such parts are used to characterize the assessment or achievement of a related goal. Altogether, the questions specify which specific aspect of the object of study are observed to understand if the goal is achieved or not. Questions are measurable entities that establish a link between the object of study and the focus. For example, if the object of study is a car and the focus is its environmental impact, a question could be: "How high are the carbon dioxide emissions of the car?".
3. **Quantitative** level (metric): defines the set of software measurements needed to answer the questions in an objective (quantitative) way.

The definition of the first level, the goal level, is critical to the successful application of the GQM approach [5]. Every measurement goal has to be described stating the purpose of the measurement (what and why it is measured), the perspective (what specifically is observed, the focus, and from which point of view the observation is made), and the environment (in which context the measurement takes place).

To ease the definition of measurement goals, the GQM supplies a template how to define a goal. The GQM goal template suggests to formulate a goal using the following sentence [16]: "Analyze (the object of study) to (the purpose) with respect to (the focus) from the point of view of (the viewpoint), in the following context (the environment)." The elements to replace in the template are explained in Table 1.

In Fig. 2, we provide an example of a GQM-model to measure the usability of a web site. The goal is defined according to the GQM goal template. We derived three questions from this goal: how efficiently the web site can be used, how intuitive the design is, and how satisfied the users are in general. The metrics connected to the questions are the time needed to accomplish pre-defined tasks (we assume one could define some standard tasks and experiment with test users how long they take to accomplish the tasks), the time until an average users can work without asking for help, and a personal evaluation of the overall satisfaction with the web site.

Table 1 GQM goal template elements (adapted from [16])

Object of the study	The object that we want to measure
Purpose	The reason why the object is measured, e.g., to characterize, evaluate, predict, motivate, control, improve, etc
Focus	Which particular aspect of the object of study, which we study, e.g., cost, correctness, defect removal, changes, reliability, user friendliness, etc
Viewpoint	From which point of view we measure, e.g., the user, customer, manager, developer, corporation, etc
Environment	Which particular aspects of the environment influence the measurement, e.g., problem factors, people factors, resource factors, process factors, etc

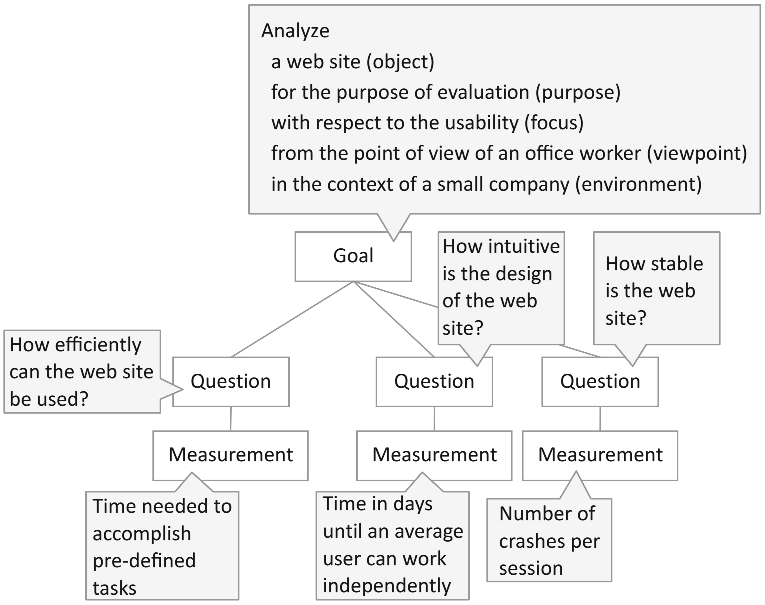

Analyze
a web site (object)
for the purpose of evaluation (purpose)
with respect to the usability (focus)
from the point of view of an office worker (viewpoint)
in the context of a small company (environment)
Goal
How intuitive is the design of the web site?
How stable is the web site?
How efficiently can the web site be used?
Question
Question
Question
Measurement
Measurement
Measurement
Time needed to accomplish pre-defined tasks
Time in days until an average user can work independently
Number of crashes per session

Fig. 2 A GQM-model to measure the usability of a web site

Once we defined which measurements we need (the quantitative level of the GQM), we can proceed with the data collection. In this paper we focus on software development process data, specifically, the software creation and the software usage process. We present the adopted approaches separately in the following two subsections.

Table 2 Examples of data collected to measure the software creation process

Timestamp	Subject	Predicate	Object
1472392020000	Developer 1	created	Test case “testA()” in class “com.company.tests.Tests”
1472478420000	Developer 4	deleted	Method “void write()” in class “com.company.Export”
1472489220000	Tester 2	executed	Test case “testB()” in class “com.company.tests.Tests”

2.1 Measuring the Software Creation Process

The goal of this step is to describe the software *creation* process so that is provides the required level of insight specified by the business objectives of the organization, i.e., what was defined in the GQM measurement goals. The goal of the measurement step is to obtain data that describes how source code is created, modified, and deleted. The data format is in subject-predicate-object triples, some examples are reported in Table 2.

Such data can be used to answer a variety of process related questions, e.g.:

- How much time are we investing in writing test cases?
- How much time are we wasting fixing defective code?
- How much time are we spending developing new features?
- How much time are we spending updating old features?
- How much do our developers collaborate in writing code? Is it one that develops each feature alone or is code modified by more people before it is released?
- Which source code was not modified since years?

Such questions represent the “know-how” of Zeleny’s taxonomy, i.e., information. To obtain the “know-why”, i.e., wisdom, one has to study information and discover or infer causalities. For example, a question could be: “Why are we slower than the competition in the development of new features?” Such a question can only be answered measuring what costs time during the development of features and to assess whether the duration of the measured activities are considered adequate or not. Those activities that take longer than the competition answer the “why” part of the question before.

To collect the data listed in Table 1, we implemented plugins for the editors, the programmers were using, e.g., Microsoft Visual Studio[1] or Eclipse.[2] These plugins track the time a developer is spending on a specific method, class, or file and send this data to a central server.

The mentioned editors provide events that are invoked whenever the developer changes his cursor position. Whenever we detect that the current method, class, file,

[1]Microsoft Visual Studio, https://www.visualstudio.com/.

[2]Eclipse IDE, https://eclipse.org/.

etc. changed, we log this change to a log file together with the timestamp. This approach limits the type and amount of data that can be collected: the data collection process should not decrease the perceived performance of the editor, otherwise the developer will not accept having such a tool installed on his or her machine.

The difficulty to implement such plugins depends from the programmability of the used editors and the quality of their documentation. One challenge is to transfer all the data to a central location, where it can be analyzed. If developers work on laptops, the data transfer component has to be able to store the data until a working Internet connection is established.

2.2 Measuring the Software Usage Process

The goal of this step is to describe the software *usage* process so that is provides the required level of insight specified by the business objectives of the organization, i.e., what was defined in the GQM measurement goals. The goal of the measurement step is to obtain data that describes how source code is used by the user. Some examples of data collected in this step are reported in Table 3.

Such data can be used to answer a variety of process related questions, e.g.:

- How many steps does it take on average for the user to perform a task?
- Which steps does the user perform to achieve typical tasks?
- Which steps may be merged because the user executes them very often?
- Which features are used very rarely or sporadically?
- Which features are hard to reach (i.e., the user needs to execute many steps to perform a specific action)?

Moreover, there are questions that can only be answered *combining* data about the software creation process with data about the software usage process, e.g., if it was worth to invest to develop a specific feature (comparing the development costs with the usage intensity).

The technologies used to collect this data depend on the type of user interface and the possibilities offered by the libraries used for programming the user interface. Some examples are:

Table 3 Examples of data collected to measure the software usage process

Timestamp	Subject	Predicate	Object
1472392020000	User 1	Clicked on	Button “Login”
1472478420000	User 2	Opened	File “/data/documents/cv.rtf”
1472489220000	User 3	Entered	Text “ajanes”

- In Microsoft .NET,[3] using the Microsoft Windows specific graphical subsystem for rendering user interfaces Windows Presentation Foundation (WPF), it is possible to define "behaviours", i.e., reusable components that can be attached to user interface elements of a given type.
- Using Aspect Oriented Programming [17] it is also possible to define "aspects", i.e., reusable components (so called "cross-cutting concerns") that can be "inserted" into various source code elements using predefined rules (the Aspect Oriented Programming community calls this step "aspect weaving").
- In web programming, monitoring is an established method to determine the performance of the currently running application and to identify failures [18, 19]. To be able to detect how software behaves on the client side, the shipped software contains a monitoring component that logs every action the user performs and records how the software reacts [19]. The client-server model of the web makes it straightforward to setup a monitoring component on the server side and to observe when, how frequent, with which parameters, from which location, etc. services are called. Nowadays, also approaches based on JavaScript injection or installing an agent on the client side are used to monitor how users interact with web sites [18].
- A relatively new approach to monitor software usage without modifying the source code of stand alone applications is to exploit the accessibility API now present in every modern operating system: operating systems like Windows, Mac OS, and Linux provide an accessibility API (e.g., [20, 21]), which allows a program to detect over which button the user positions the mouse cursor, which window he opens, which tab she selects, and so on. The original purpose of this API was to support the creation of software that helps blind users or users with bad eyesight to operate a computer. Typically, such software reads the currently focused element aloud using text-to-speech technologies. In our case, we use this API to construct a tool that—without any modification to the original source code—is able to log every user interaction with the monitored software.

Also in this case, as mentioned in Sect. 2.1, the instrumented software should not be particularly slowed down by the presence of the measurement component.

The outcome of the measurement step is data. These data have to be used to justify the costs of their collection. We report some examples of analyzing such data in the next section.

3 Analysis

In part, the choice of how to analyze the collected data was already done in defining the GQM models: the collected data serves to answer the GQM-questions.

A first approach to use the collected data is to inform all team members of the outcome of the measurement. To help all team members to focus on the information

[3]Microsoft .NET Framework, https://www.microsoft.com/net.

they need, we visualize the collected data in form of dashboards, which are visualized on large monitors in the rooms where developers work. Developers interested in the details behind the dashboard, can download a detailed report from the server.

The idea to place the dashboard in a prominent place originates from an Agile practice, the "informative workplace", which recommends to arrange the work space in a way that an "interested observer is able to walk into the team space and get a general idea of how the project is going in fifteen seconds" [22]. Together with other tools, e.g., a Kanban board that shows which tasks are scheduled, which are currently in progress, and which are done, we also show a dashboard, visualizing metrics that help the team to keep an eye on key aspects of the development.

To design the dashboards, we follow the guidelines by Stephen Few [23], i.e., to use dashboards as "visual displays of the most information needed to achieve one or more objectives which fits entirely on a single computer screen so it can be monitored at a glance." Since we use GQM models to define which data we want to collect and why we collect it, we design the dashboards after the defined GQM models. An example of such a dashboard is shown in Fig. 3.

Such a dashboard visualizes the obtained values for the chosen measurements in form of bar charts, if the obtained value has to be considered "good", "average", or "poor" within the bar chart, using different background shadings, and the trend of the value of the past using sparklines, i.e., small line charts that visualize the general shape of the variation over time. This, because sometimes the exact value of a metric is not important but the trend is. For example, it can be relatively unimportant to know that the number of open defects is 57, while it is important to know that it increased by 100% from yesterday.

If an obtained value has to be considered "good", "average", or "poor" is defined through rules, that state between which ranges values fall into each category. These rules can be either obtained from the literature, estimated, or based on past data.

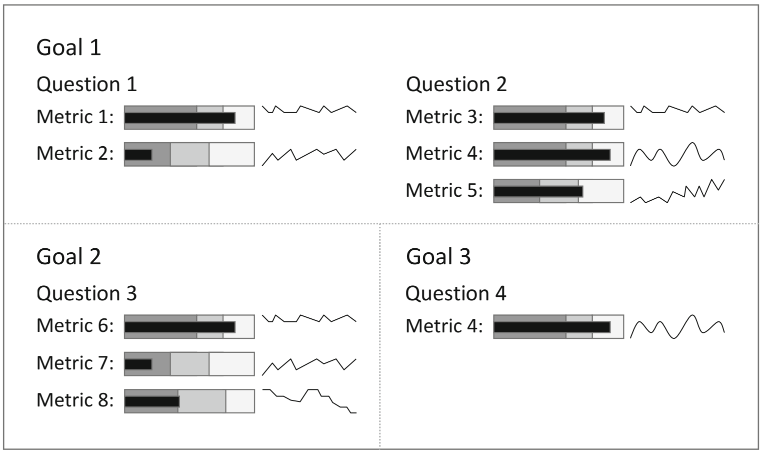


Fig. 3 An example dashboard to visualize metrics structured according to GQM models

The different metrics are organized according to the questions their answer and the goals to which the questions belong. The so visualized data becomes information, because we show its context, using the GQM-approach.

In the following two sub-sections we describe two examples that go beyond the mere visualization of the collected information: the generation of wisdom, i.e., the analysis of the "know-why" of Zeleny's taxonomy. We describe two cases, one studying the software creation process, one studying the software usage process.

3.1 Analyzing the Software Creation Process

In the first case, we studied how a software development team is investing its time within small software company. The company management felt that there is a trade-off between two aspects: exploration or exploitation [5]:

- **Exploitation**: with exploitation the company management understood the improvement of existing features of their software. This includes the fixing of defects as well as the enhancement of existing features. Exploitation, in this case, helps to maintain the existing customer base, but, as the competition is constantly improving their products, on the long run is not enough to ensure that the company remains profitable.
- **Exploration**: with this term the company management understood the innovation of their products, the implementation of new features to address new requirements by existing customers but particularly potential customers that could not be acquired so far because of missing features.

Exploitation and exploration are both important: exploitation is needed to keep existing customers (which generates yearly maintenance fees), exploration is needed to gain new customers. Too much exploitation (also called "gold-plating" a product) is not useful to acquire new customers in new markets, too much exploration disappoints existing customers since their products are not maintained and motivates them to switch to the competition.

The goal for the management of the company was to study the software creation process, precisely, to measure exploitation and exploration.

The GQM developed for this measurement project was as follows:

- **Goal**: Analyze the software development process for the purpose to balance with respect to the time developers spend writing code from the point of view of the software developer in the context of a software development project
- **Questions**:
  1. Which artifacts are developers working at?
  2. How much time do they work at the different artifacts?

Table 4 Examples of data collected to measure the amount of exploration and exploitation

Timestamp	Duration (minutes)	Subject	Predicate	Object
1472392020000	10	Developer 1	Created	Method "testA()" in class "com.company.tests.Tests"in file "Tests.java"
1472478420000	2	Developer 4	Edited	Method "void write()" in class "com.company.Export" in file "Export.java"
1472489220000	7	Tester 2	Edited	Method "testB()" in class "com.company.tests.Tests" in file "Tests.java"

- **Metrics**:
  1. For question 1 we extract properties useful to identify the artifact:
     - the file name of the focused artifact,
     - the package name of the focused artifact,
     - the class name of the focused artifact, and
     - the method signature of the focused artifact.
  2. For question 2:
     - the time spent working on the focused artifact.

The outcome of the measurement is data similar to the one described in Sect. 2.1, shown in Table 4.

Once the time spent on each source code part was logged, the next step was to classify it as "exploitation" or "innovation". We decided to—instead of asking the developer—infer it based on the creation date of the file in which the source code was modified. The justification for this heuristics was that the architecture of the application was such that the various features were coded as separate files. New features would be coded in new files to keep the maintainability of the entire code base high. Older features are contained in files created longer time ago and exploitation means that the developers have to edit files that exist in the repository already for some time.

Therefore, to calculate the amount of exploration (innovation), we are interested in how much time was spend on new files. The newer the file, the more likely it is that the modification was an implementation of something new. Following the same logic, if the file is old, we do not want to count the editing time as much as for new files.

To achieve this, we multiply all editing times with $\gamma_{exploration}^{\beta}$, where $\gamma_{exploration}$ is between 0 and 1 and is the attenuation factor for exploration, a number that expresses how much we want to reduce the time if the file is old and β a number that expresses the age of the file. Therefore, the degree of **exploration** of a developer d that edited n files is given by:

$$d_{exploration} = \sum_{f=0}^{n} effort_d(f) \times \gamma_{exploration}^{age(f)},$$

where the function $effort_d(f)$ returns the time a user spent editing the file f and $age(f)$ returns the age the file f. In our case we calculated the effort of a file in minutes, the age of a file in months, and chose $\gamma = 0.5$.

Accordingly, the degree of **exploitation** is calculated as follows:

$$d_{exploitation} = \sum_{f=0}^{n} effort_d(f) \times \gamma_{exploitation}^{age(f)},$$

where $0 < \gamma_{exploration} < 1$ is the attenuation factor for exploitation. By choosing $\gamma_{exploitation} = (1 - \gamma_{exploration})$, the sum $d_{exploration} + d_{exploitation}$ corresponds to the actual effort the user spent editing source code files and the values of $d_{exploration}$ and $d_{exploitation}$ correspond to a distribution of the effort collected by our measurement probes, weighted by the age of the files that were edited.

The resulting values for $d_{exploration}$ and $d_{exploitation}$ are estimates that help the company to understand how much importance is given to exploration or exploitation.

Setting up a system as the one described here is not straightforward. Some of the issues we encountered are discussed in Sect. 4.

3.2 Analyzing the Software Usage Process

In the second case, we studied how users use a software product within a small company to better understand how users interact with software to learn how the user interface could be improved [24].

The objective of this study was to observe users that interact with an internally developed software. The goal for the management of the company was to improve the user interface: based on the activities the users perform, frequently used features (or feature combinations) can be simplified or merged to increase the efficiency of the user. Less frequently used features can be removed or the user can be reminded that those features exist. Unexpected interactions can help to identify missing features, e.g., we observed that some users used the field "birth place" to store data for which no field existed in the software.

The GQM developed for this measurement project was as follows:

- **Goal**: Analyze the internally developed software for the purpose to evaluate the usability from the point of view of the user in the context of a small company.
- **Questions**:

  1. How many steps does it take on average for the user to perform a task?
  2. Which steps does the user perform to carry out typical tasks?

Table 5 Examples of data collected to measure the usability of a software product

Timestamp	Subject	Predicate	Object
1472392020000	User 1	Opened	Application
1472478420000	User 1	Opened	Window "Orders"
1472489220000	User 1	Clicked on	Order "2016/125"
1472489220000	User 1	Clicked on	Button "Print"
1472489220000	User 1	Closed	Window "Order 2016/125"
1472489220000	User 1	Clicked on	Order "2016/187"

3. Which steps may be merged because the user executes them very often?
4. Which features are used very rare or sporadically?
5. Which features are hard to reach (i.e., the user needs to execute many steps to perform a specific action)?

- **Metrics**:

1. To answer all questions we log every interaction with user interface elements the users perform, e.g., clicks on buttons, text editing in text boxes, etc.

The outcome of the measurement is data similar to the one described in Sect. 2.2, shown in Table 5.

Since we wanted to study the sequence of steps a user typically executes to accomplish typical tasks, we decided to use process mining to extract sequences of events from the collected data.

The term "process mining" describes a set of techniques that aim to study processes by extracting knowledge from process traces, i.e., logs that document events that happened because a process was executed [25].

In a first step, a process model is generated using the event log (this step is called "process discovery"). The so "discovered" or "mined" process model can be used to compare it with an ideal process model (this activity is called "conformance checking") or/and to support process improvement, i.e., to study processes to find bottlenecks, unexpected delays, observe the distribution of work, etc.

In our case, we used the generated process model to answer the questions stated in the GQM-model.

Our initial idea to use process mining was to discover the underlying process that users are using in interacting with the software. In doing this, we faced process mining-specific difficulty: process mining algorithms expect to get a set of cases as input, not just a set of events. This means that the flow of events has to be divided into "chunks", i.e., "cases" that have a fundamental impact on the outcome of the algorithm: process mining algorithms look for similarities and differences between cases. In many situations—as in ours—deciding when a case begins and when it ends is not a trivial decision.

Table 6 Examples of data collected to measure the usability of a software product, including a case-ID

Timestamp	Case ID	Subject	Predicate	Object
1472392020000	1	User 1	Opened	Application
1472478420000	1	User 1	Opened	Window "Orders"
1472489220000	1	User 1	Clicked on	Order "2016/125"
1472489220000	1	User 1	Clicked on	Button "Print"
1472489220000	1	User 1	Closed	Window "Order 2016/125"
1472489220000	2	User 1	Clicked on	Order "2016/187"
1472489220000	2	User 1	Clicked on	Button "Edit"
1472489220000	2	User 1	Modified	Textbox "Delivery date"

We considered the following possibilities:

1. One day corresponds to one case. This means that we set a new case ID per user and per day. With this assignment of the case ID, a case has many events and we can observe the interaction of a user throughout the day.
2. We assign a new case ID every time a user starts the application or comes back to the main form. In this way we obtain shorter and more repetitive cases that reflect how a user navigates through the software.
3. We assign a new case ID every time a user selects some specific item (e.g., an order, a customer, a product) in the software. Such a perspective allows to observe how users work when they deal with a specific item.

For this study we choose option two, since we wanted to understand how the user interacts with the program during typical usage scenarios, which typically start selecting a record in the main form and to operate on the selected record in subsequent, nested windows. Whenever the user went back to the main form, it was to select another record and to perform a different action.

This choice led to the data as shown in Table 6. Whenever the user comes back to the main window "Orders", we increase the case ID by one.

We visualizing this log using the process mining software Disco,[4] which uses the Fuzzy Mining algorithm [26]. The resulting process models depict how users typically navigate through the software. The extracted, simplified, process model is shown in Fig. 4. To reduce the complexity of the extracted process model, Fig. 4 shows only 5% of the paths and only 10% of the activities. This is why some paths are depicted in gray, this means that there are hidden activities behind the the path, and some are depicted in black, which means that there are no hidden activities.

We then used the obtained process models to answer the questions stated in the GQM.

To **determine how many steps it takes on average for the user to perform a task**, we need to define what we mean by a task. In our case carrying out a task means

[4]Disco, https://fluxicon.com/disco/.

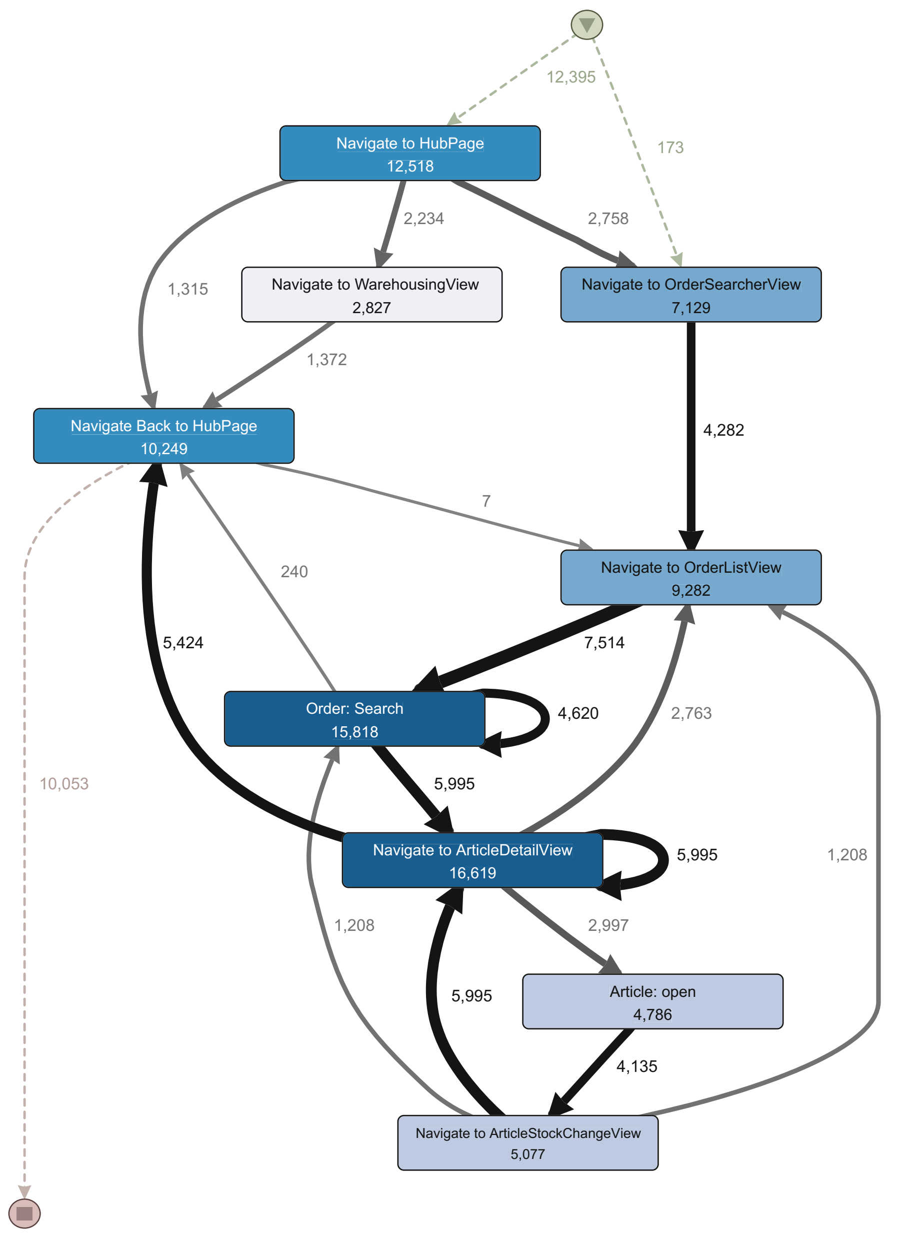


Fig. 4 Process model extracted from the logged user interactions (visualizing 10% of the activities and 5% of the paths)

to use the program and come back to the main screen. To answer this question we can define interesting events in the process model, e.g., "START EDIT Article", and count the steps needed to reach this functionality.

To answer the question **which steps the user performs to carry out typical tasks** we can also use the process map to see which specific steps a user performs.

The same process map can also be used to answer the question **which steps may be merged because the user executes them very often**. In the example shown in Fig. 4, the program flow currently forces the user to open the order form ("START OrderAdministration") to see order details. Users then typically just close the order form (they invoked "CANCEL OrderAdministration" in 232 from 292 cases). A possible improvement would be to allow the user to see some order details (e.g., the delivery date) in the previous window, the order list.

To find the **features that are used very rarely or sporadically** we can define a filter so that the process map is constructed only for events that occur below a defined threshold.

To identify **features that are hard to reach (i.e., where the user needs to execute many steps to perform a specific action)** we can also use the process map to identify tasks with many events.

In this sub-section we studied how users interact with software, while in the previous sub-section we looked at how programmers create software. Through the combination of both aspects, interesting questions about software development can be answered, e.g., looking at how much a certain feature took to develop we can measure if it is used by the users, and how often it is used. In this way we can estimate the benefit for the users and how much the investment paid off for the company.

Nevertheless, one has to be careful in this consideration: it is not necessarily so that frequently used features are the most important for the user and vice versa that the less used feature are the least important. It might be that a feature is used frequently because there is no other way to obtain a certain result.

4 Discussion

In the previous sections, we described the motivation for developing systems that help to collect data, information, knowledge and wisdom about software.

It was our priority to measure without distraction, which has the several advantages but also drawbacks, that we discuss in this section.

Non-distracting, automatic, data collection has the advantage that those that are measured can focus on their main task, that the data can be collected continuously, and that data is more precise than when collected manually.

On the other hand, usually nobody wants that data is collected in the background without knowing exactly which data are collected, who will receive that data, and how that data will be used. Without trust, the described data collection method can have strong negative effects: there is the risk that constant observation is perceived as intimidating, felt as a permanent stress factor, and lead to an overall lower performance [27].

Being constantly observed can be seen as a decrease in autonomy, i.e., that the available alternatives for taking action that one has [28, 29] diminish. A decrease in autonomy, i.e., feeling that one has not the freedom to choose what is best for him, has a negative impact on the motivation, which furthermore can have negative consequences in the productivity of the single employee [30, 31].

Productivity in software development is highly correlated on the motivation of the involved people. People that have a personal interest in what they are doing are far more productive than those that are not [32]. Therefore, companies are interested to keep their motivated programmers, not to scare them away, and to help them to stay productive.

Therefore, we distinguish two possible uses of measurement: to enhance understanding or to control. If an input to the developers is experienced as an information, it enhances their autonomy, whereas if it is experienced as controlling it diminishes autonomy [29].

To use measurement to enhance understanding and autonomy, the collected data has to be used with the clear goal to improve the overall performance of the team, not to identify who is guilty of particular problems. The defined GQM-models can help in this context to communicate to everybody *what* will be collected, *why* it will be collected, and *how* the data will be interpreted.

Moreover, we suggest that the collected data can be visualized, modified or deleted manually after it was sent to the server by everybody. This has several advantages: first, it anticipates trust among those that are measured and those that use the collected data, second, those that are measured do not need to fear that data is collected that disclose something that they do not want to disclose, and third, if the collected data does not reflect the reality, e.g., data was lost because of a hardware failure, one can correct the mistake manually.

The derived conclusions on how to improve the software development process or how to design a new version of the analyzed application have to be validated with developers and users to ensure that the collected data were not misinterpreted. Possible conclusions that can be drawn from the collected data about the software development process are:

- Based on the collected data about how the development team spends its time, the team might decide that it needs more training to perform certain activities or that it needs to invest in automating laborious tasks that do not provide value if carried out manually.
- Based on the collected data about the frequency and type of modifications on source code, the team might decide that certain source code might be discontinued, refactored, or extended.
- Based on the collected data about the amount of rework and the time spent on fixing defective code, the team might decide to change the requirement engineering process, e.g., investing in techniques that allow the user to better envision the final solution, or the team might also decide to change the software development

approach modularizing the created software in a different way, e.g., using micro-services, to increase the modifiability of the created solution.

Possible conclusions that can be drawn from the collected data about the software usage process are:

- Based on the steps users need to take to perform a given task, the team might decide to simplify the application, splitting it into two applications: one to accomplish day-to-day tasks, and one to configure more complex aspects of the software.
- Based on the frequency features are used, the team might decide to remove, merge, or improve features. Particularly features that are heavily used are indicators of functionality that the user values: according to [33], the perceived value is closely related to usage and the consequences of usage. In the case of software, this means that parts of the software that are used more often are indicators of value. The frequency of use of a particular feature can be used by the software development team to prioritize the testing effort: features that are extensively used should be extensively tested.
- Based on the changes in the usage of features over time or in combination with other software, the team can understand if the user is experiencing problems and can react on such changes timely.

As can be seen by these examples, using logged data to infer knowledge and wisdom has the advantage that the collected data reflects traces that real events left behind, i.e., it is an indication of the events that really (not ideally) happened. A drawback is that instead of having the possibility to ask directly *how* or *why* somebody did something, we have to laboriously collect data about occurred events and often use heuristics or assumptions to guess or infer *how* or *why* an artifact has been created, modified, or deleted.

5 Conclusion

This chapter presented a practical approach to obtain data, information, knowledge, and wisdom (using Zeleny's terminology) within software engineering. "Practical" since knowledge management initiatives often fail when companies "lack time and resources to focus on the meaning, implications, capturing and sharing of organizational knowledge management [34]."

An approach that does not distract the most valuable resources (developers and users) lowers the costs of collecting data but not the costs of studying and using it. Through the public display of dashboards, the GQM-approach, and the open availability and changeability of the collected data within the software development team, we promote the open discussion within all stakeholders whether the displayed information is useful, how it should be used, if it should be changed, or removed.

References

1. Brooks Jr., F.P.: No silver bullet essence and accidents of software engineering. Computer **20**(4), 10–19 (1987)
2. Royal Academy of Engineering and British Computer Society: The Challenges of Complex IT Projects: The Report of a Working Group from the Royal Academy of Engineering and the British Computer Society. The Royal Academy of Engineering (2004), accessed 30 Sept 2016. http://www.bcs.org/upload/pdf/complexity.pdf
3. Zeleny, M.: Management support systems: towards integrated knowledge management. Human Syst. Manag. **7**(1), 59–70 (1987)
4. Ackoff, R.L.: From data to wisdom. J. Appl. Syst. Anal. **16**, 3–9 (1989)
5. Janes, A., Succi, G.: Lean Software Development in Action. Springer, Berlin (2014)
6. Meiran, N.: Reconfiguration of processing mode prior to task performance. J. Exp. Psychol.: Learn. Memory Cognit. **22**(6), 1423–1442 (1996)
7. Monsell, S., Yeung, N., Azuma, R.: Reconfiguration of task-set: is it easier to switch to the weaker task? Psychol. Res. **63**, 250–264 (2000)
8. Rogers, R.D., Monsell, S.: Costs of a predictable switch between simple cognitive tasks. J. Exp. Psychol.: General **124**, 207–231 (1995)
9. Rubinstein, J.S., Meyer, D.E., Evans, J.E.: Executive control of cognitive processes in task switching. J. Exp. Psychol. Hum. Percept. Perform. **27**(4), 763–797 (2001)
10. Johnson, P.M., Kou, H., Agustin, J., Chan, C., Moore, C., Miglani, J., Zhen, S., Doane, W.E.J.: Beyond the personal software process: metrics collection and analysis for the differently disciplined. In: Proceedings of the International Conference on Software Engineering (ICSE). IEEE Computer Society, Portland, Oregon (2003)
11. Ikonen, M.: Leadership in kanban software development projects: A quasi-controlled experiment. In: Abrahamsson, P., Oza, N.V. (eds.) Proceedings of the International Conference on Lean Enterprise Software and Systems (LESS). Lecture Notes in Business Information Processing, vol. 65. Springer, Helsinki, Finland (2010)
12. Czerwinski, M., Horvitz, E., Wilhite, S.: A diary study of task switching and interruptions. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI). ACM, Vienna, Austria (2004)
13. van Solingen, R., Berghout, E., van Latum, F.: Interrupts: just a minute never is. IEEE Softw. **15**(5), 97–103 (1998)
14. Jenkins, S.: Concerning interruptions. IEEE Comput. **39**(11) (2006)
15. Basili, V.R., Caldiera, G., Rombach, H.D.: The goal question metric approach. In: Marciniak, J.J. (ed.) Encyclopedia of Software Engineering, vol. 1. Wiley (1994)
16. Basili, V.R., Trendowicz, A., Kowalczyk, M., Heidrich, J., Seaman, C., Münch, J., Rombach, D.: Aligning Organizations Through Measurement: The GQM+Strategies Approach. The Fraunhofer IESE Series on Software and Systems Engineering. Springer International Publishing, Berlin (2014)
17. Kiczales, G.: Aspect-oriented programming. ACM Comput. Surv. **28**(4es), 148 (1996)
18. Croll, A., Power, S.: Complete Web Monitoring: Watching your Visitors, Performance, Communities, and Competitors. O'Reilly Media, Sebastopol (2009)
19. Thalheim, B., Schewe, K., Prinz, A., Buchberger, B.: Correct Software in Web Applications and Web Services. Texts & Monographs in Symbolic Computation. Springer International Publishing, Berlin (2015)
20. Microsoft: .NET Framework Development Guide, UI Automation Overview (2016). https://msdn.microsoft.com/en-us/library/ms747327(v=vs.110).aspx
21. Apple: Accessibility Programming Guide for OS X (2016). https://developer.apple.com/library/mac/documentation/Accessibility/Conceptual/AccessibilityMacOSX/
22. Beck, K.: Extreme Programming Explained: Embrace Change, 2nd edn. Addison-Wesley, Boston (2004)
23. Few, S.: Information Dashboard Design: The Effective Visual Communication of Data. Oreilly Series. O'Reilly Media, Sebastopol (2006)

24. Astromskis, S., Janes, A., Mairegger, M.: A process mining approach to measure how users interact with software: an industrial case study. In: Proceedings of the 2015 International Conference on Software and System Process, ICSSP 2015, pp. 137–141. ACM, New York, NY, USA (2015)
25. van der Aalst, W.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer, Berlin (2011)
26. Günther, C.W., Aalst, W.M.P.v.d.: Fuzzy Mining — Adaptive Process Simplification Based on Multi-perspective Metrics. In: Business Process Management, Lecture Notes in Computer Science, vol. 4714. Springer (2007)
27. Brödner, P., Knuth, M.: Nachhaltige Arbeitsgestaltung: Trendreports zur Entwicklung und Nutzung von Humanressourcen. Bilanzierung innovativer Arbeitsgestaltung, Hampp (2002)
28. Mikl-Horke, G.: Industrie- und Arbeitssoziologie, 3 edn. Oldenbourg Wissenschaftsverlag (1995)
29. Deci, E.L., Connell, J.P., Ryan, R.M.: Self-determination in a work organization. J. Appl. Psychol. **74**(4), 580 (1989)
30. Deci, E.L., Ryan, R.M.: The "what" and "why" of goal pursuits: Human needs and the self-determination of behavior. Psychol. Inquiry **11**(4), 227–268 (2000)
31. Deci, E.L., Ryan, R.M., Gagne, M., Leone, D., Usunov, J., Kornazheva, B.: Need satisfaction, motivation, and well-being in the work organizations of a former eastern bloc country a cross-cultural study of self-determination. Personal. Soc. Psychol. Bull. **27**(8), 930–942 (2001)
32. Herzberg, F.: Decoding the dna of the toyota production system. Harv. Bus. Rev. **46**(1) (1968)
33. Woodruff, R.B.: Customer value: The next source for competitive advantage. J. Acad. Market. Sci. **25**(2), 239–153 (1997)
34. Chan, I., Chao, C.K.: Knowledge management in small and medium-sized enterprises. Commun. ACM **51**(4), 83–88 (2008)