Chapter 6
Semantic Matching of Engineering Data Structures

Olga Kovalenko and Jérôme Euzenat

Abstract An important element of implementing a data integration solution in multi-disciplinary engineering settings, consists in identifying and defining relations between the different engineering data models and data sets that need to be integrated. The ontology matching field investigates methods and tools for discovering relations between semantic data sources and representing them. In this chapter, we look at ontology matching issues in the context of integrating engineering knowledge. We first discuss what types of relations typically occur between engineering objects in multi-disciplinary engineering environments taking a use case in the power plant engineering domain as a running example. We then overview available technologies for mappings definition between ontologies, focusing on those currently most widely used in practice and briefly discuss their capabilities for mapping representation and potential processing. Finally, we illustrate how mappings in the sample project in power plant engineering domain can be generated from the definitions in the Expressive and Declarative Ontology Alignment Language (EDOAL).

Keywords Ontology matching · Correspondence · Alignment · Mapping · Ontology integration · Data transformation · Complex correspondences · Ontology mapping languages · Procedural and declarative languages · EDOAL

6.1 Introduction

Ontology and data matching tasks are key steps for semantic data integration (Breslin et al. 2010) and therefore they often surface in real-world applications that have the need to exploit relations between the concepts and instances of two data sources.

O. Kovalenko (✉)
Institute of Software Technology and Interactive Systems, CDL-Flex,
Vienna University of Technology, Vienna, Austria
e-mail: olga.kovalenko@tuwien.ac.at

J. Euzenat
INRIA & Univ. Grenoble Alpes, Grenoble, France
e-mail: jerome.euzenat@inria.fr

Ontology alignments express these relations explicitly in order to be processed as mappings for various applications, e.g., for data transformation or query rewriting (Shvaiko and Euzenat 2013).

Although much work has been done on ontology schema matching and defining data-level mappings, it is not obvious, especially for non Semantic Web experts, which of the provided tools and technologies will be well-suited in a specific real-life application case and how to apply them (Vyatkin 2013). For engineering practitioners, it might be challenging to understand (a) which technologies can be used to represent the relations between their data models and data sets; and (b) how to use these for a specific application. On the other hand, Semantic Web researchers might lack insight on (1) what are the needs (in terms of mappings) of real-world applications in engineering; and (2) how well existing mapping technologies can support defining such mappings. In this chapter, we aim to address these gaps for the benefit of practitioners and Semantic Web researchers alike. Therefore, we formulated the following *research questions* for the chapter: (Q1) What kind of relations are expected to occur often between the engineering data models and data sets? (Q2) What are the available mapping languages developed by Semantic Web community and what are their characteristics? (Q3) How well the existing mapping languages can support defining and representing the relations identified for Q1?

In order to answer Q1 we synthesized our experiences from (a) implementing semantic integration in the real-life application scenario from our industry partner, a power plant systems integrator; and (b) literature analysis in the fields of ontology matching and schema matching. As a result, we derive a catalog of schema and data correspondences that are expected to arise often in ontology mediation applications (see Sect. 6.4 for details). We focus on complex correspondence types that are not obvious with respect to their representation, i.e., those that go beyond simple one-to-one mapping (like `owl:sameAs`) between the entities. These are already well-explored and supported by existing ontology matching solutions. For more information about existing matching techniques, systems, and tools please see a comprehensive review by (Otero-Cerdeira et al. 2015) or (Euzenat and Shvaiko 2013).

Our application scenario, a power plant engineering project, belongs to the automation system engineering domain where mappings between ontologies are used to support data integration. The intended application in this case is enabling data transformation from the local ontologies of the different engineering disciplines involved in a project into a common ontology in order to allow the project-level integration (see Sect. 6.3 for the detailed description). As in the presented use case ontology mapping is used for data transformation, we will use it as default application for the sake of coherence across the chapter. Therefore, when speaking about mappings in this chapter, it can be assumed that they are done for data transformation.

In Sect. 6.5 we overview available technologies for mappings definition, representation and processing and briefly discuss their main usages in current practice and potential strengths and limitations w.r.t. representing correspondences identified in Sect. 6.4 (therefore, addressing Q2 and Q3). Finally, Sect. 6.6 introduces the EDOAL language, gives the examples of how the identified correspondence types can be implemented in this language and shows how they can be used for data transformation. We chose EDOAL because, contrary to other languages, it combines the features of both declarative and procedural languages: (a) it has been designed for expressing correspondences (conforms to declarative languages); (b) but at the same time those correspondences can be interpreted to perform linking, i.e., primitive Silk scripts (Volz et al. 2009), SPARQL[1] queries, SPARQL+SPIN[2] can be generated to perform data transformation for instance.

Our main contributions therefore are the following:

1. A catalog of complex correspondence types with the examples of those in real-world application scenario in the automation systems engineering domain;
2. A succinct overview of existing mapping languages, their principal features and capabilities;
3. A detailed analysis of EDOAL capabilities to support identified correspondences types.

6.2 Ontology Matching: Background Information and Definitions

In this section we provide background information on ontology matching and give the definitions of the important terms, which will be used across the chapter. We will follow the terminology from the "Ontology Matching" book (Euzenat and Shvaiko 2013) and from (Scharffe et al. 2014):

Ontology matching is the process of finding relations between the entities of different ontologies, e.g., identifying the entities that represent the same (or similar) semantics in these ontologies.

Correspondence is the expression of a relation holding between entities (classes, properties, or individuals) of different ontologies. Correspondences express the essence of how the entities are related independently from any application or implementation details in a specific mapping language.

Alignment is a set of correspondences between two ontologies. The alignment is the output of the ontology matching process.

Mapping specifies the connection between ontology entities in enough details to process or execute them for a certain task. Execution in this case means using a specific engine/tool that can read the mapping, understand its semantics and run it according to the intended application. The output of the execution will vary depending on specific application. For instance, for data transformation it will be data set conforming to the target ontology; and for query rewriting the initial query (formulated in the source ontology vocabulary) will be transformed into the query formulated according to the target ontology vocabulary. A mapping, therefore, is a correspondence expressed in a specific mapping language with a certain exploitation in mind.

[1] https://www.w3.org/TR/sparql11-query/.

[2] http://www.w3.org/Submission/spin-overview/.

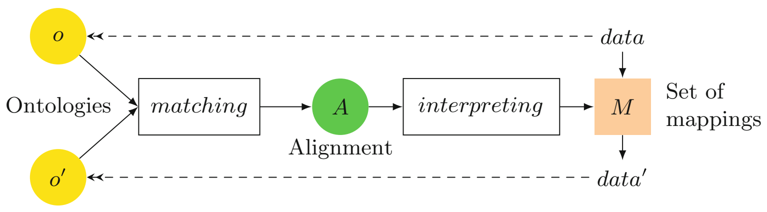

Fig. 6.1 Ontology matching for data transformation. Ontology matching generates an alignment A between ontology o and o'. This alignment can be interpreted as a set of mappings M, which can be used for transforming data expressed in ontology o into ontology o'

Both correspondences and mappings can be specified on the data (ontology instances) or schema (ontology classes and properties) level.

The main terms defined above as well as possible articulation between these terms (for the data transformation scenario) are illustrated in Fig. 6.1.

Three main dimensions may be distinguished in the ontology matching area (Noy 2004): (a) alignments discovery, i.e., ontology matching, (b) alignments representation, and (c) applying them as mappings.

The majority of works in the area is dedicated to *ontology matching*, which is a challenging and complex problem. Although many tools have been developed using different approaches, e.g., language-based, structure-based, or context-based, discovery processes tend to be highly iterative and cannot be fully automated in the majority of cases (Bernstein and Melnik 2007). Interested reader can find the detailed description of various alignments discovery approaches in (Euzenat and Shvaiko 2013).

Another important issue in the ontology matching area is the *application of mappings*, i.e., to be applied for a specific task alignments should be implemented in some mapping language; and then the obtained mappings can be processed for an application at hand. The precise type of application will for a large part determine the choice of a language to express mappings. Typically applications aim to reduce semantic heterogeneity. The often reported applications are ontology merging, data translation, and ontology integration.

Finally, the third dimension of the ontology matching area is dedicated to *defining and representing* alignments and mappings. Different languages may be used for that purpose with various characteristics (see Sect. 6.5 for more details). At the same time, there is a lack of guidelines (especially for non Semantic Web experts) on how to select a language for specific application and what are the important features of these languages that will influence the choice.

In this chapter we focus on the representation problem, in particular when this involves complex relations between entities. As we see in next sections, this is the case in the engineering domain.

6.3 Running Example: The Power Plant Engineering Project

Power plant engineering projects represent complex environments where participants from different disciplines (e.g., mechanical, electrical, and software engineering), have to collaborate to deliver high quality end products while satisfying tight timeframes (Mordinyi et al. 2011).

This application scenario is based on a case study implemented in a power plant engineering project of our industry partner, an automation system solution provider. Initially, different disciplines within the project applied isolated engineering processes and workflows: (a) isolated data sets stored in various proprietary data formats and discipline-specific relational databases; and (b) tools were loosely coupled with limited capabilities for cross-disciplinary data exchange or/and data analysis. The goal of the case study was to implement an ontology-based integration solution in order to allow cross-disciplinary data exchange and data analysis for advanced applications, such as change propagation and consistency checking across the discipline boundaries.

After analyzing the project requirements the hybrid ontology approach for integration according to (Wache et al. 2001) (see Fig. 6.2) has been chosen for the final system. According to this approach, a local ontology is built for each engineering discipline, gathering discipline-specific data models, knowledge, and constraints, and finally, data. Then an additional integrating layer is introduced—a common knowledge base—that contains only concepts, which are important on the project level. These concepts are called *common concepts* (Biffl et al. 2011), because they are typically relevant within at least two engineering disciplines. For more detailed description of common concepts please see Chap. 4 *“The Engineering Knowledge Base Approach.”* The final system can be implemented following one of the two basic approaches: (a) the CC ontology defines the structure of the common knowledge base and data is stored in database(s); or (b) the CC ontology is populated with

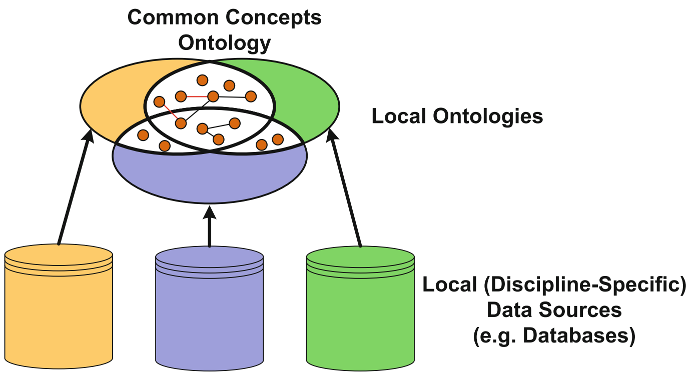

Fig. 6.2 Hybrid ontology integration (adapted from (Wache et al. 2001))

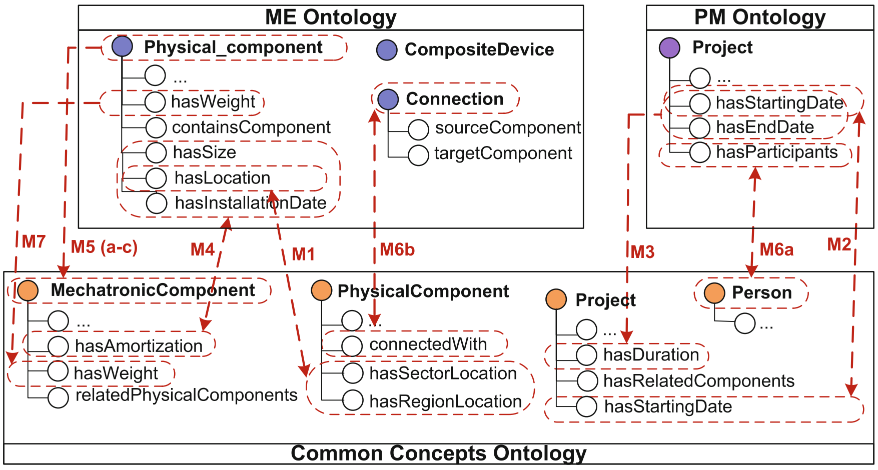

Fig. 6.3 Ontology-based integration in the power plant engineering project

instances and serves itself as a common knowledge base for a project. In both cases the advantage is that the CC ontology defines a common vocabulary on the top of the engineering project and provides a single access point for data analysis across the disciplines. Within the case study the second approach was chosen. In the application scenario, mappings are defined between the local discipline-specific ontologies and the CC ontology to serve as a basis for data transformation from the local level into the CC ontology.

We present the part of the ontological system developed for the case study as the running example in this chapter. In this example, the two domains are integrated: (a) *mechanical engineering* (ME), responsible for designing the physical structure of devices and connections between them, and (b) *project management* (PM), responsible for managing the information about past and current projects, and people involved in the development. To construct and populate the CC ontology, it is necessary to transform data from the domain models into the integrated model according to specified mappings between the domain ontologies and the CC ontology.

Figure 6.3 illustrates the constructed ontological system. Each domain is represented by its local ontology. The ME ontology comprises entities related to the physical topology of a power plant, and the PM ontology includes entities related to personnel involved in the development and project organization aspects. The CC ontology includes only those entities that are relevant on the project level. For the sake of simplicity, the set of objects and properties (shown in the running example) is limited to the minimal set necessary to illustrate all the correspondences types that are introduced further in Sect. 6.4 (see Fig. 6.3). These correspondences specify the various relations between the entities of local ontologies and the CC ontology and, when implemented as mappings, can define data transformations from the local storages to the integrated storage.

6.4 Representing Relations Between Engineering Objects

In this section, we overview what kind of correspondences and mappings between ontology entities may occur while capturing relations between the engineering data models and data (though the described ones are not strictly specific to the engineering domain and may arise in other domains as well).

In order to identify the presented correspondences we followed a bottom-up approach. First, we analyzed what types of relations occur often in the engineering data, summarizing our experiences with implementing semantic integration in the multi-disciplinary engineering projects of the industry partner. Second, we performed a literature analysis in the ontology mediation, ontology matching, and schema matching fields, in order to verify that identified correspondences are not specific to our scenarios, but are indeed widespread and recognized by researchers and practitioners from different domains and can be found in a wide range of application scenarios. Besides describing the correspondences in general, we provide concrete examples of those from the application scenario presented in Sect. 6.3. To better position the identified correspondences in the ontology matching field we link them to the ontology alignment design patterns[3] and the work of F. Scharffe on correspondence patterns (Scharffe 2009).

The presented correspondences can be expected to occur frequently in various ontology mediation tasks. At the same time, they require the use of an expressive formalism in order to be properly defined. We proceed with the detailed description of these correspondences in general and examples for those from the real-life application scenario.

Value Processing (M1–M4)
Often, the relation between values of two entities can be represented by some function that takes a value on one side as an input and returns a value on another side as an output, i.e., some processing is needed to map the entities. The complexity of this processing varies from simple string operations to sophisticated mathematical functions. This type of correspondences is considered in (Scharffe 2009) as the "Property value transformation" pattern, where the author distinguishes between the string-based, operations on numbers and unit conversion transformations. In the following, several types of such correspondences are described in detail.

String Processing (M1)
Description. As string values are widely used for data representation, the processing of string values is often needed while transforming data from one ontology into another. Expressing such correspondences requires using special functions on string values, e.g., "concat," "substring," or "regex." An example of this correspondence for the "concat" function can be found in (Scharffe 2009) under the name of "Class Attribute to Attribute Concatenation Pattern."

Example. In the ME ontology each physical component's location is defined via the `hasLocation` property, whose value is a string combining sector and region

[3]http://ontologydesignpatterns.org/wiki/Category:AlignmentOP.

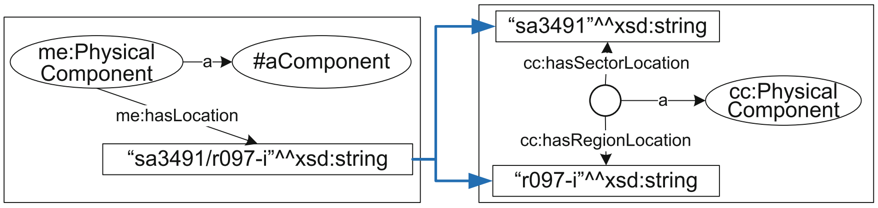

Fig. 6.4 M1: String processing correspondence example

(defines location within a specific sector) information for a specific component. In the CC ontology the location information of a physical component is explicitly divided into two separate properties. The correspondence specifies that the initial string must be split into two parts, which then will be used to construct values for the `hasSectorLocation` and the `hasRegionLocation` properties in the CC ontology (see Fig. 6.4).

Data Type Transformation (M2)
Description. It can happen that in a proprietary data model the data type of a certain property was not modeled in an optimal way, e.g., the date can be represented in the format of "string," instead of "Date" or "DateTime." This type of correspondence encodes, therefore, how the value of one type in source ontology can be transformed into value of a different type in a target ontology.

Generally, the data types can be compatible, partially compatible or incompatible (Legler and Naumann 2007). For instance, "integer" and "string" are compatible data types (although only uni-directionally); "date" and "string" are partially compatible (the compatibility will depend on a specific value); and "DateTime" and "integer" are incompatible data types. For the compatible data types there is also a possibility to specify correspondence in a more general way, i.e., specifying how these data types can be transformed into each other. For the example above it could be defining how any "Date" value should be transformed into a "string" value. In this case, the inverse mapping cannot be defined in a general way.

Example. All values of the `hasStartingDate` property in the PM ontology are strings in the following format "DD/MM/YYYY." But because the data type of the corresponding property in the CC ontology is "Date," a data type transformation correspondence takes place between these two properties (see Fig. 6.5).

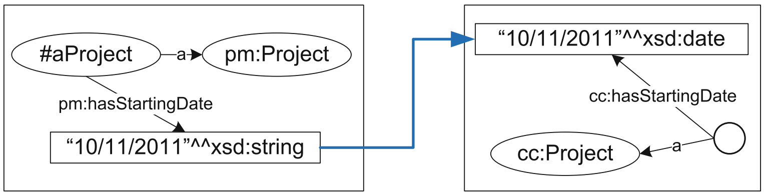

Fig. 6.5 M2: Transforming a string into xsd:date

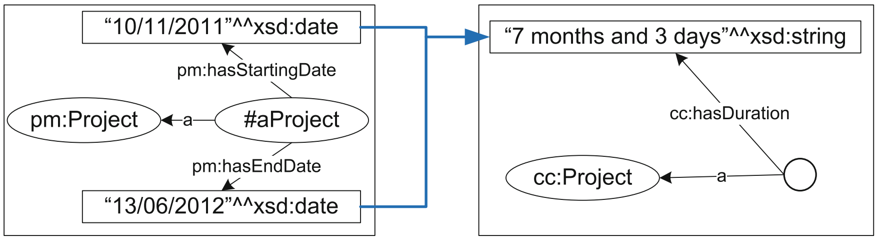

Fig. 6.6 M3: Computing the duration of a project

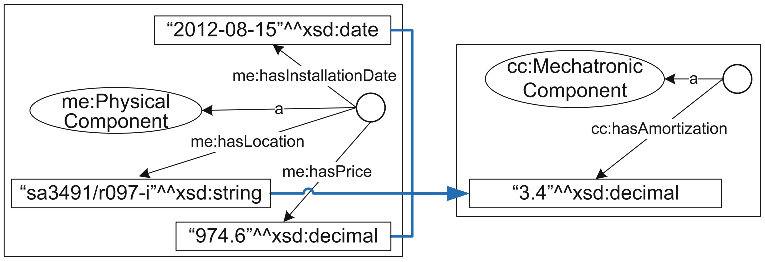

Fig. 6.7 M4: Computing the amortization of a component based on its properties

Math Functions (M3)
Description. In this case value processing involves some mathematical operations or is specified by a formula representing mathematical, physical, or other natural laws. This, for an instance, can be such simple mathematical operations as addition or multiplication, or more complex functions such as finding an integral or logarithm. However, relation capturing in this case is done by the means of the used mapping language.

Example. The value of the `hasDuration` property in the CC ontology is equal to substracting `hasStartingDate` from `hasEndingDate` in the PM ontology (see Fig. 6.6).

User-defined Functions (M4)
Description. For this type of correspondences, the relation is expressed by functions that are not supported by the used mapping language/technology, but must be additionally implemented. Therefore, it must be possible to call an external function, e.g., implemented in Java, that will generate a property value or an entity in a target ontology.

Example. The concept `MechatronicComponent` in the CC ontology captures information regarding complex composite components, which consist of many physical components and basically can represent a plant itself. The anticipated amortization value can be an important characteristic for such objects. The exact value will depend on the location, price and installation date of a specific mechatronic component (see Fig. 6.7).

Structural Differences (M5–M6)
A frequent situation is that two ontologies, covering the same or similar knowledge area, were designed by different people and in different time, following different modeling approaches and not aware of each other. In this case, the same semantics can be modeled very differently. This type of correspondences serves to smooth such kind of differences.

Granularity (M5)
Description. In this case, the same real-life object is modeled at a different level of detail in the two ontologies.

Example. In the ME ontology, the concept `Physical_component` is used to represent both single devices, e.g., a specific sensor, and complex objects that comprise many devices, e.g., a part of a production plant or the plant itself. In the CC ontology, there are two objects that distinguish between composite and single devices, i.e., a single device is represented by `PhysicalComponent` and composite objects are represented by `MechatronicComponent`. To encode this connection between the ME and CC ontologies, one has to properly classify specific physical components in ME ontology. This is usually done by encoding a specific conditioning into the defined correspondence.

For instance, for the presented example one can perform filtering of the mechatronic components based on property value, i.e., saying that those physical components, which weight more than a specific threshold, are mechatronic components (see Fig. 6.8).

Another option could be to filter based on property occurence, i.e., saying that mechatronic components are those physical components that contain other devices. To check that one can use the existense of `containsComponent` property for a specific physical component in the ME ontology (see Fig. 6.9).

One more example of filtering could be checking whether a physical component also belongs to a specific type, e.g., saying that mechatronic components are those physical components that are of type `compositeDevice` in the ME ontology (see Fig. 6.10).

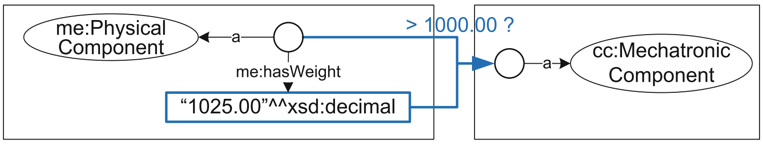

Fig. 6.8 M5a: Defining mechatronic components by property value

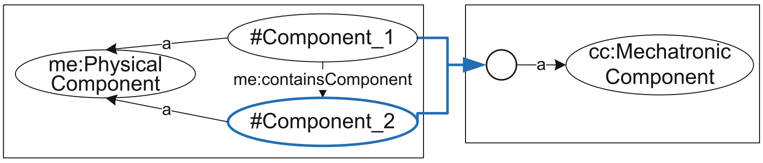

Fig. 6.9 M5b: Defining mechatronic component by property occurence

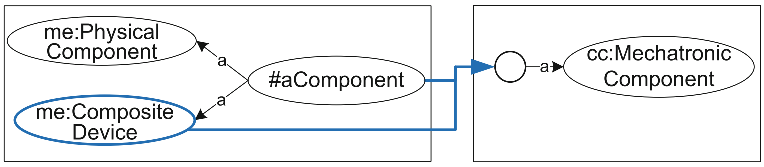

Fig. 6.10 M5c: Defining mechatronic component by instance type

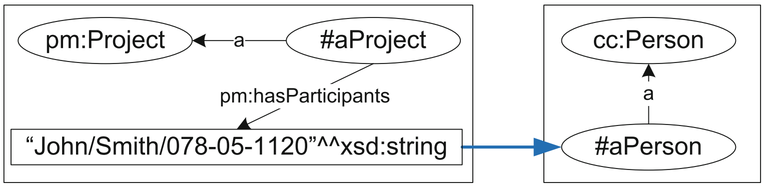

Fig. 6.11 M6a: Correspondence between the property value in ME ontology and instance in CC ontology

Similar types of correspondences and examples for them are reffered in (Scharffe 2009) as the "Class Correspondence by Path attribute Value," "Property Correspondence by Value," "Class by Attribute Occurence Correspondence' patterns. Also, similar patterns are described by the "Class_by_attribute_occurence," "Class_by_attribute_type," "Class_by_attribute_value," and "Class_by_path_attribute_value" ontology design paterns.[4]

Schematic Differences (M6)
Description. In this case, there are substantial differences in the way the same semantics is represented in the two ontologies.

Example1. Each employee in the PM ontology is represented by a string value of the `hasParticipants` property, while in the CC ontology the concept `Person` serves the same purpose. The correspondence captures this relation between a property value and a class instance (see Fig. 6.11).

Example2. A connection between physical devices in the ME ontology is represented by the `Connection` concept with the `sourceComponent` and `target Component` properties, while in the CC ontology the same semantics is expressed with the `connectedWith` property of the `PhysicalComponent` concept (see Fig. 6.12).

The correspondences with similar semantics and corresponding examples can be found in (Scharffe 2009) denoted as the "Class Relation Correspondence," "Property–Relation Correspondence," and "Class Instance Correspondence" and also within the ontology design patterns ("Class correspondence defined by relation domain").

[4]Mentioned design patterns can be found under http://ontologydesignpatterns.org/wiki/Submissions:#pattern_name.

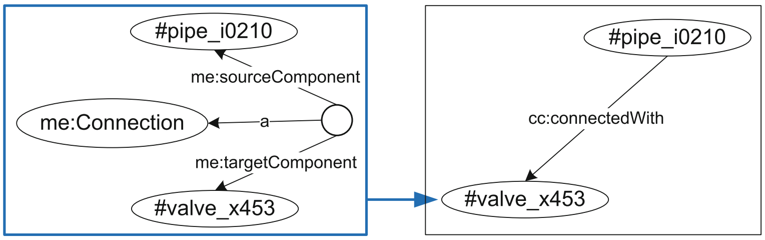

Fig. 6.12 M6b: Correspondence between class in ME ontology and property in CC ontology

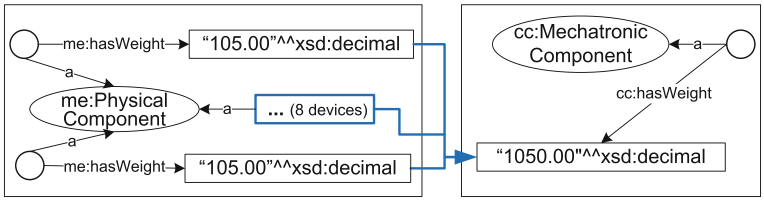

Fig. 6.13 M7: Aggregation of property values to get the weight of a mechatronic device

Grouping and Aggregation (M7)
Description. In some cases it is important to use grouping or/and aggregation of entities in one or several ontologies in order to set the relation to another ontology. This type of correspondence is also presented in (Scharffe 2009) as the "Aggregation" pattern.

Example. In order to calculate a value of the property `hasWeight` for a specific `MechatronicComponent` in the CC ontology, values of the `hasWeight` property of all devices from the ME ontology, which are contained in this component, should be summed up (see Fig. 6.13).

Mapping Directionality
When speaking about mappings, an important characteristic is that they can be directional (Ghidini et al. 2007), i.e., can be specified in a direction from source to target and the data flow cannot occur in the opposite direction. However, for some applications, such as for a data transformation, it could be beneficial to define bidirectional mappings between the engineering objects. It would help to reduce the total amount of mappings, thus facilitating their maintenance. However, in some cases, it may be impossible to specify a bidirectional mapping—e.g., in the example for mapping type M3 it will not be possible to specify the specific values for start and end dates based only on the duration of a specific project.

Example. Examples for M1 and M2 (if specified in a specific mapping language) can also serve as examples of bidirectional mappings.

6.5 Languages and Technologies for Mapping Definition and Representation

This section provides a description of languages and technologies that can be applied for ontology mapping. Even though many initiatives exist to map heterogeneous sources of data to RDF such as XSPARQL (Akhtar et al. 2008) to transform XML into RDF and RML (Dimou et al. 2014) to map CSV, spread sheets and XML to RDF, we will only examine those languages that allow expressing alignments and mappings between different ontologies.

Although the languages described below are of very different nature for the sake of uniformity hereafter we will call them "mapping languages." For this chapter, we focus on those languages which are already well known and/or widely used. This means that these languages already have implementations and tool support and, therefore, would be the most probable and convenient choice for practitioners.

All mapping languages can be divided into the two categories: declarative and procedural languages. A language is *declarative*, if it expresses something independently from the way it is processed. Therefore, one should use external tools to process the defined correspondences for an application at hand. A *procedural* language, on the other hand, expresses how mappings are processed (for a specific or various applications).

Another important characteristic of a mapping language is whether it is suited to express correspondences at *schema* (classes and properties) or *data* (ontology instances) level. Below we provide a brief description of existing mapping languages. Table 6.1 position them with respect to these categories.

The **Web Ontology Language (OWL)** is an ontology language where one can declare relations between concepts such as equivalence, subsumption, etc., and allows one to infer additional information about instances by reasoning over the properties of classes and relations. Although one can define one-to-one correspondence

Table 6.1 Mapping languages and their characteristics: ● = compliant; ○ = non compliant

	Declarative	Procedural	Schema	Data
OWL	●	○	●	●
SWRL	●	○	○	●
SPARQL CONSTRUCT	○	●	○	●
Jena rules	○	●	○	●
SPIN	○	●	○	●
SILK	○	●	○	●
SKOS	●	○	●	●
SEKT	●	●	●	●
Alignment format	●	○	●	●
EDOAL	●	●	●	●

between the ontology entities using `owl:equivalentClass`, `owl:equivalentProperty` and `owl:sameAs` for classes, properties and individuals correspondingly, OWL itself has no means to define more complex correspondences as those described in Sect. 6.4. Also, as OWL is a knowledge representation language, it by itself possesses no means for data transformation between ontologies. One will need to use additional tools for that, such as OWL reasoners to infer additional triples[5] from an OWL file. Thus, reasoners could be used in combination with SPARQL CONSTRUCT queries to create a "reasoner-enabled" mapping.

The **Semantic Web Rule Language, (SWRL)**[6] is a W3C submission for a Semantic Web rule language, combining OWL DL—a decidable fragment of OWL—with the Rule Markup Language.[7] Rules are thus expressed in terms of OWL concepts. Rules are of the form of an implication between an antecedent (body) and consequent (head). The intended meaning is: whenever the conditions specified in the antecedent hold, then the conditions specified in the consequent must also hold. Note that SWRL rules are not intended for defining mappings, but to infer additional information from an ontological system, i.e., if the intended application is instance translating, they should be used in combination with SPARQL CONSTRUCT queries for instance to create a target RDF graph out of a source graph.

One way to define mappings is to use a **SPARQL CONSTRUCT**[8] query, which returns an RDF graph created with a template for generating RDF triples based on the results of matching the graph pattern of the query. To use this construct, one needs to specify how patterns in one RDF graph are translated into another graph. The outcome of a CONSTRUCT query depends on the reasoner and rule engine used. A SPARQL endpoint not backed by an OWL reasoner will only do simple graph matching for returning triples. A software agent that needs to compute these inferences will therefore have to consume all the necessary triples and perform this computation itself. The same holds for inferring additional information via business rules. SPARQL CONSTRUCT, however, is not a rule language and "merely" allows one to make a transformation from one graph match to another graph, i.e., a one-step transformation.

Another option to define mappings is using rules that can be declared on top of OWL ontologies. Apache Jena[9] includes a rule-based inference engine called **Jena Rules**[10] for reasoning with RDF and OWL data sources based on a Datalog implementation. Datalog is a declarative logic programming language that is popular for data integration tasks (Huang et al. 2011).

SPARQL Inference Notation (SPIN)[11] is currently submitted to W3C and provides—amongst others—means to link class definitions with SPARQL queries

[5]RDF triple: https://www.w3.org/TR/2004/REC-rdf-concepts-20040210/#section-triples.

[6]http://www.w3.org/Submission/SWRL/.

[7]http://wiki.ruleml.org/index.php/RuleML_Home.

[8]http://www.w3.org/TR/rdf-sparql-query/#construct.

[9]Apache Jena: http://jena.apache.org/.

[10]https://jena.apache.org/documentation/inference/#rules.

[11]http://www.w3.org/Submission/spin-overview/.

(ASK and CONSTRUCT) to infer triples. An implementation of SPIN is available from TopBraid.[12] It is built on top of the Apache Jena framework, and therefore inherits its properties.

Silk (Volz et al. 2009) is a link discovery framework for RDF data sets available as a file or via SPARQL endpoints. It allows one to declare how different RDF data sets relate to each other by specifying so-called linkage rules. These linkage rules are used to identify which resources are related to generate, for instance, owl:sameAs predicates. One is also able to define correspondences using other predicates, which depend on the use case. These linkage rules can make use of aggregates, string metrics, etc. SILK allows describing how resources in two existing data sets relate to each other, but does not possess means to process them for a certain application, e.g., to perform the transformation.

Two systems for expressing relations between entities worth mentioning are **SKOS** (Miles et al. 2005) and the **Alignment format** (David et al. 2011). However, they can only express correspondences between pairs of named entities of two ontologies, so they are not suited to address the requirements in Sect. 6.4.

Another language to define mappings was developed by the **SEKT** project. This language is designed to be independent from any ontology language, thus, it can be used for ontologies written in different languages. Several syntaxes are available—verbose human readable syntax and RDF and XML syntaxes. A Java API is also available allowing parsing and serializing to and from the object model of the mapping document. This mapping language is quite expressive—it allows specifying mappings between classes, properties, and instances of an ontology (also across) using a set of operators, which have a cardinality, an effect and some related semantics. One can also specify conditions, annotations, direction info (bidirectional or unidirectional mapping) and extend the defined constructs with arbitrary logical expressions (Scharffe et al. 2006).

The **EDOAL**[13] **(Expressive and Declarative Ontology Alignment Language)** (David et al. 2011) is a language for expressing alignments which is supported by the Alignment API. The Alignment API allows to generate and parse alignments, to manipulate them and to render these alignments in different languages, eventually executable. EDOAL can express correspondences between more precise and complex terms than the named entities. EDOAL also supports expressing transformations on property values, which is of particular interest in our context.

Table 6.2 summarizes the level of support of the mapping languages listed above for defining and representing complex relations between the ontologies described in Sect. 6.4. The evaluation was done based on (a) checking the specification documents for each language; (b) authors' practical experiences with implementing ontology-based integration solutions; and c) knowledge obtained during authors' involvement in the language development (for some languages).

Due to space limits we cannot provide a detailed analysis for each of the described mapping languages. From Table 6.2, it is clear that everything can be written directly

[12]http://www.topquadrant.com/.

[13]http://alignapi.gforge.inria.fr/edoal.html.

Table 6.2 Support for the complex relations definition and representation in various mapping languages: ●–supported; ◐–partially supported; ○– no support; *–vendor dependent

	M1	M2	M3	M4	M5a	M5b	M5c	M6a	M6b	M7
SWRL	◐	●	●	◐	●	●	●	●	●	●
SPARQL CONSTRUCT	●	●	●	●	●	●	●	●	●	●
Jena rules	◐	◐	◐	● *	●	●	●	●	●	●
SPIN	●	●	●	● *	●	●	●	●	●	●
SILK	●	●	●	●	●	●	●	○	●	◐
SKOS	○	○	○	○	○	○	○	○	○	○
SEKT	●	●	●	●	●	●	●	○	●	○
Alignment Format	○	○	○	○	○	○	○	○	○	○
EDOAL	●	●	●	●	●	●	●	○	●	●

with SWRL or SPARQL CONSTRUCT. Such languages have enough expressivity and can be considered, at least for SPARQL, to have efficient implementations. However, they lack declarativity. For instance, they define oriented rules and changing the orientation requires rewriting the rules. A language like EDOAL allows to express declaratively the relations between two ontologies and can generate SPARQL CONSTRUCT (or eventually SWRL in simple cases) to implement the transformations in one way or the other. Therefore, we decided to focus on EDOAL. We continue with detailed analysis of the EDOAL's capabilities for representing complex correspondences and the examples of those identified in the use case scenario (see Sect. 6.3).

6.6 Representing Complex Relations with EDOAL

Expressive and Declarative Ontology Alignment Language (EDOAL) is an extension of the Alignment format supported by the Alignment API (David et al. 2011). It offers the capability to express correspondences that go beyound putting in relation named entities. In EDOAL correspondences may be defined between compound descriptions, which allow to further constrain those entities that are put in correspondences. Compound descriptions may be combination of concepts, e.g., a physical component that is also a composite device, or restriction of concepts, e.g., a physical component whose weight is over 125 pounds. Compound descriptions are defined through a description language similar to that of description logics.

This is possible through:

Construction that constrains the object put in correspondence with classical Boolean operators (disjunction, conjunction, complement) or property construction operators (inverse, composition, reflexive, transitive, and symmetric closures);

Restriction that restrains the values of objects put in correspondence (through domain, range, cardinality and value restrictions). This is typically achieved by requesting all or some values of a property to be in some domain (`weight` is above 1025) or class (some member being a senior manager).

These constraints can be composed together to obtain more elaborate correspondences. For instance, the meaning of some of the correspondences in Sect. 6.4 can be simply expressed as EDOAL correspondences.

The snippet in Listing 6.1 combines M5a and M5c examples in one correspondence (M5b can be treated in the exact same way). The first part (within the "`and`") is a class construction, while the `AttributeValueRestriction` part is a class restriction.

Listing 6.1 EDOAL correspondence capturing M5a and M5c examples

```
<align:Cell rdf:about="#M5">
  <align:entity1>
    <Class>
      <and rdf:parseType="Collection">
        <Class rdf:about="&me;PhysicalComponent"/>
        <Class rdf:about="&me;CompositeDevice"/>
         <AttributeValueRestriction>
           <onAttribute><Relation rdf:about="&me;hasWeight"/></
               onAttribute>
           <comparator rdf:resource="&edoal;greater-than"/>
           <value><Literal edoal:type="&xsd;decimal"edoal:string
               ="1025.00"/></value>
         </AttributeValueRestriction>
      </and>
    </Class>
  </align:entity1>
  <align:entity2>
    <Class rdf:about="&cc;MechatronicComponent"/>
  </align:entity2>
  <align:relation>=</align:relation>
  <align:measure rdf:datatype="&xsd;float">1.0</align:measure>
</align:Cell>
```

The Alignment API offers EDOAL alignment manipulation and rendering. So, the correspondence presented in Listing 6.1 can be automatically transformed into the SPARQL CONSTRUCT query shown in Listing 6.2, which can be used for transforming data.

Listing 6.2 SPARQL CONSTRUCT query generated from the EDOAL correspondence in Listing 6.1

```
CONSTRUCT { ?s rdf:type cc:MechatronicComponent . }
WHERE {
  ?s rdf:type me:PhysicalComponent .
  ?s rdf:type me:CompositeDevice .
  ?s me:hasWeight ?w .
  FILTER( ?w >="1025.00"^^xsd:decimal ).
}
```

The same can be achieved with the M6b correspondence example that connects an object and a relation between objects (see Listing 6.3).

Listing 6.3 EDOAL correspondence capturing M6b example

```
<align:Cell rdf:about="#M6b">
  <align:entity1>
    <Relation>
      <compose rdf:parseType="Collection">
        <Relation>
          <inverse><Relation rdf:about="&me;sourceComponent"/></
              inverse>
        </Relation>
        <Relation>
          <and rdf:parseType="Collection">
            <Relation rdf:about="&me;targetComponent"/>
            <RelationDomainRestriction>
              <class><Class rdf:about="&me;Connection"/></class>
            </RelationDomainRestriction>
          </and>
        </Relation>
      </compose>
    </Relation>
  </align:entity1>
  <align:entity2><Relation rdf:about="&cc;connectedWith"/></
      align:entity2>
  <align:relation>=</align:relation>
  <align:measure rdf:datatype="&xsd;float">1.0</align:measure>
</align:Cell>
```

In addition to expressing the usual correspondences between two entities, EDOAL correspondences are extended to support two types of information:

Transformations that specify how to transform an instance of the related entities into the other;

Link keys that define under which conditions two individuals must be considered the same (Atencia et al. 2014).

For this chapter we are mostly concerned with transformations. They allow expressing how property values of two equivalent relations can be transformed into one another. For that purpose EDOAL usually applies transformation functions on the values.

This is, in general, sufficient to express the examples for M1–M4 and M7. In Listing 6.4 the correspondence covering M2 and M3 (where M2 is a simple conversion from string to date; and M3 is a function call though the call of substract-dates) is presented. In general, all M1–M4 examples would follow the same pattern.

Listing 6.4 EDOAL correspondence capturing M2 and M3 examples

```
<align:Cell rdf:about="#M1extended">
  <align:entity1><Class rdf:about="&pm;Project"/></align:entity1
      >
  <align:entity2><Class rdf:about="&cc;Project"/></align:entity2
      >
  <align:relation>=</align:relation>
  <align:measure rdf:datatype="&xsd;float">1.0</align:measure>
  <transformation>
    <Transformation edoal:direction="o-">
      <entity1>
            <Apply edoal:operator="xsd:date">
         <arguments rdf:parseType="Collection">
            <Property rdf:about="&pm;hasStartingDate"/>
```

```
          </arguments>
        </Apply>
      </entity1>
      <entity2><Property rdf:about="&cc;hasStartingDate"/></
          entity2>
    </Transformation>
    <Transformation edoal:direction="o-">
      <entity1>
       <Apply edoal:operator="op:subtract-dates">
         <arguments rdf:parseType="Collection">
           <Property rdf:about="&pm;hasStartingDate"/>
           <Property rdf:about="&pm;hasEndDate"/>
         </arguments>
       </Apply>
      </entity1>
      <entity2><Property rdf:about="&cc;hasDuration"/></entity2>
    </Transformation>
  </transformation>
</align:Cell>
```

As it can be seen in the snippet in Listing 6.4, the same EDOAL correspondence can support several transformations as, for two objects in the correspondence, there may be several properties to be transformed.

Listing 6.5 EDOAL correspondence capturing M7 example

```
<align:Cell rdf:about="#M1extended">
  <align:entity1><Class rdf:about="&me;PhysicalComponent"/></align
      :entity1>
  <align:entity2><Class rdf:about="&cc;MechatronicComponent"/></
      align:entity2>
  <align:relation>=</align:relation>
  <align:measure rdf:datatype="&xsd;float">1.0</align:measure>
   <transformation>
    <Transformation edoal:direction="-o">
      <entity1>
            <Aggregate edoal:operator="&fn;sum">
             <arguments rdf:parseType="Collection">
            <Property>
              <compose rdf:parseType="Collection">
                <Relation rdf:about="&me;component"/>
                <Property rdf:about="&me;hasWeight"/>
              </compose>
            </Property>
            </arguments>
          </Aggregate>
      </entity1>
      <entity2><Property rdf:about="&cc;hasWeight" /></entity2>
    </Transformation>
  </transformation>
</align:Cell>
```

The example for M7 can be expressed in a similar way (see Listing 6.5) Alternatively, the transformation here could have been added to the previous M5 example.

With regard to directionality, EDOAL correspondences are in general not oriented: they express a relation between two terms and not a function from one term to another. However, when generating a mapping from a correspondence, this may direct the use of the correspondence, e.g., the SPARQL query above. Moreover, transformations may be oriented. This is specified by the direction attribute: indeed, it is possible to compute a duration from the dates or a total weight of the compound component from its component part weights but not the other way around.

Although EDOAL can express such transformations, it cannot process them. These transformations are rendered in other languages that can be processed, such as SPARQL. Transformations often rely on external functions, most of the time identified by XPath URIs. Hence, a renderer using them should be able to interpret such operations as SPIN functions or SPARQL built-in functions, for instance.

One type of correspondences that is difficult to represent in EDOAL, are those, like M6a, which introduce some new resources that have no counterpart in the initial data set. In M6a, the `Person` in the CC ontology does correspond to `John/Smith/078-05-1120`, which is a simple string. Sometimes this can be easily expressed, like for M6b, because the `connectedWith` relation relates two existing resources. But in general, this requires more information.

6.7 Conclusion

In this chapter, we described what types of complex relations between engineering objects may need to be captured, while implementing ontology-based integration solutions. We presented a catalogue of complex correspondences between the ontologies and illustrated each correspondence type with an example from the real-life power-plant engineering project. We also provided an overview on available mapping languages and technologies that can be used to define, represent, and sometimes also process the correspondences between the different ontologies and briefly explained their key characteristics and capabilities w.r.t. the complex relations capturing. As space limitations do not allow us to perform the detailed analysis of each mapping language, we decided to focus on EDOAL, because of its interesting quality to combine the features of both declarative and procedural languages (contrary to other mapping languages). We therefore explained in detail what are the EDOAL's capabilities for representing complex relations and gave the examples of how the correspondences identified for the power-plant engineering project can be implemented with EDOAL.

As future work we would like to explore what languages and techniques are currently applied by engineers to link different models and data structures (especially across the engineering disciplines and tools) during the engineering process and to compare their representational capabilities with those provided by the Semantic Web community. Another interesting direction for the future work will be analyzing languages to define relations and constraints on different data models and data sets developed within the Model-Driven engineering field and compare their application aspects with the languages discussed in this chapter.

Acknowledgments This work was supported by the Christian Doppler Forschungsgesellschaft, the Federal Ministry of Economy, Family and Youth, and the National Foundation for Research, Technology and Development in Austria.

References

Akhtar, W., Kopeckỳ, J., Krennwallner, T., Polleres, A.: XSPARQL: Traveling Between the XML and RDF Worlds—and Avoiding the XSLT Pilgrimage. Springer (2008)

Atencia, M., David, J., Euzenat, J.: Data interlinking through robust linkkey extraction. In: Proceeding 21st European Conference on Artificial Intelligence (ECAI), Praha (CZ), pp. 15–20 (2014)

Bernstein, P.A., Melnik, S.: Model management 2.0: manipulating richer mappings. In: Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data, pp. 1–12. ACM (2007)

Biffl, S., Moser, T., Winkler, D.: Risk assessment in multi-disciplinary (software+) engineering projects. Int. J. Softw. Eng. Knowl. Eng. **21**(02), 211–236 (2011)

Breslin, J.G., O'Sullivan, D., Passant, A., Vasiliu, L.: Semantic web computing in industry. Comput. Ind. **61**(8), 729–741 (2010)

David, J., Euzenat, J., Scharffe, F., Trojahn Dos Santos, C.: The Alignment API 4.0. Semant. Web J. **2**(1), 3–10 (2011)

Dimou, A., Vander Sande, M., Colpaert, P., Verborgh, R., Mannens, E., Van de Walle, R.: RML: a generic language for integrated RDF mappings of heterogeneous data. In: Proceedings of the 7th Workshop on Linked Data on the Web (LDOW2014), Seoul, Korea (2014)

Euzenat, J., Shvaiko, P.: Ontology Matching, 2nd edn. Springer, Heidelberg (DE) (2013)

Ghidini, C., Serafini, L., Tessaris, S.: On relating heterogeneous elements from different ontologies. In: Modeling and Using Context, pp. 234–247. Springer (2007)

Huang, S.S., Green, T.J., Loo, B.T.: Datalog and emerging applications: an interactive tutorial. In: Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data, pp. 1213–1216. ACM (2011)

Legler, F., Naumann, F.: A classification of schema mappings and analysis of mapping tools. BTW, Citeseer **103**, 449–464 (2007)

Miles, A., Matthews, B., Wilson, M., Brickley, D.: SKOS core: simple knowledge organisation for the web. In: International Conference on Dublin Core and Metadata Applications, p. 3 (2005)

Mordinyi, R., Winkler, D., Moser, T., Biffl, S., Sunindyo, W.D.: Engineering object change management process observation in distributed automation systems projects. In: Proceedings of the 18th EuroSPI Conference, Roskilde, Denmark (2011)

Noy, N.F.: Semantic integration: a survey of ontology-based approaches. ACM SIGMOD Rec. **33**(4), 65–70 (2004)

Otero-Cerdeira, L., Rodríguez-Martínez, F.J., Gómez-Rodríguez, A.: Ontology matching: a literature review. Exp. Syst. Appl. **42**(2), 949–971 (2015)

Scharffe, F.: Correspondence patterns representation. Ph.D. thesis, University of Innsbruck (2009)

Scharffe, F., de Bruijn, J., Foxvog, D.: Ontology mediation patterns library v2. Deliverable **D4**, 3 (2006)

Scharffe, F., Zamazal, O., Fensel, D.: Ontology alignment design patterns. Knowl. Inf. Syst. **40**(1), 1–28 (2014)

Shvaiko, P., Euzenat, J.: Ontology matching: state of the art and future challenges. IEEE Trans. Knowl. Data Eng. **25**(1), 158–176 (2013)

Volz, J., Bizer, C., Gaedke, M., Kobilarov, G.: Silk: a link discovery framework for the web of data. LDOW 538 (2009)

Vyatkin, V.: Software engineering in industrial automation: state-of-the-art review. IEEE Trans. Ind. Inf. **9**(3), 1234–1249 (2013)

Wache, H., Voegele, T., Visser, U., Stuckenschmidt, H., Schuster, G., Neumann, H., Hübner, S.: Ontology-based integration of information—A survey of existing approaches. In: IJCAI-01 Workshop: Ontologies and Information Sharing, Citeseer, vol. 2001, pp. 108–117 (2001)