Ontology Management

Article · January 2009

DOI: 10.1007/978-3-540-88845-1_2

CITATIONS
11

READS
1,394

7 authors, including:

Stephan Bloehdorn
IBM
43 PUBLICATIONS **1,588** CITATIONS
SEE PROFILE

Peter Haase
metaphacts GmbH
168 PUBLICATIONS **5,671** CITATIONS
SEE PROFILE

Zhisheng Huang
Vrije Universiteit Amsterdam
212 PUBLICATIONS **2,847** CITATIONS
SEE PROFILE

York Sure-Vetter
Karlsruhe Institute of Technology
207 PUBLICATIONS **7,944** CITATIONS
SEE PROFILE

Chapter 2
Ontology Management

Stephan Bloehdorn, Peter Haase, Zhisheng Huang, York Sure, Johanna Völker, Frank van Harmelen, and Rudi Studer

Abstract Ontologies are considered a key technology enabling semantic interoperability and integration of data and processes. We are now entering a phase of knowledge system development, in which ontologies are produced in larger numbers and exhibit greater complexity than before. Ontology management infrastructures are needed for the increasing development of semantic applications especially in the corporate semantic web, which comprises the application of semantic technologies in an enterprise environment. This chapter presents a generic system architecture for an ontology management infrastructure that supports the construction of semantic applications. Such an ontology management infrastructure needs to be generic and flexible in its design, comprehensive in its functionality in order to meet the requirements of a wide spectrum of semantic applications. Key component functionalities of an ontology management system are discussed, including ontology mapping, ontology learning, debugging reasoning with inconsistent ontologies, and (structured and unstructured) query answering. An ontology management system architecture which can flexibly build upon existing datas sources and connect to other applications and which – as such – also forms the basis for a wide range of semantically enabled applications is described.

2.1 Introduction

Ontologies are considered a key technology enabling semantic interoperability and integration of data and processes. We are now entering a phase of knowledge system development, in which ontologies are produced in larger numbers and exhibit greater complexity. Ontology management infrastructures are needed for the increasing development of semantic applications especially in the corporate semantic web, which comprises the application of semantic technologies in an enterprise environment.

S. Bloehdorn (✉)
Institute AIFB, University of Karlsruhe, D-76128 Karlsruhe, Germany
e-mail: bloehdorn@aifb.uni-karlsruhe.de

J. Davies et al. (eds.), *Semantic Knowledge Management*,

In this chapter, we present a generic system architecture for an ontology management infrastructure that supports the construction of semantic applications. Such an ontology management infrastructure is expected to be generic and flexible in its design, comprehensive in its functionality in order to meet the requirements of a wide spectrum of semantic applications. To illustrate typical requirements of a semantic application, we start with a short scenario of semantic integration and search:

Bob works as technology analyst for an IT company. His daily work includes research on new technological trends, market developments and competitors of his clients. Bob's company maintains a digital library that gives access to a repository of internal surveys and analysis documents. The company also has a license with an academic research database which is accessed via a separate interface. Depending on his work context, Bob uses the topic hierarchies, the fulltext search functionalities or metadata search facilities provided by the two libraries to get access to the relevant data. However, Bob is often annoyed by the differing topic hierarchies and metadata schemes used by the two libraries as well as by a cumbersome syntax for posing metadata queries.

Questions which Bob might be interested in could include:

1. What articles were published by William Arms in "Communications of the ACM"?
2. Who wrote the book "Digital Libraries"?
3. What article deals with Grid Computing?
4. What are the topics of the paper "The future of web services"?
5. Which conference papers were classified as "emerging technology trends 2007"?
6. Which articles are about "Intellectual Capital"?

This scenario can be seen a prototypical scenario in which ontologies may be used for an integration of corporate information sources to enable semantic search for everyday research tasks. From this initial scenario and the above questions, we derive the following requirements that need to be met by an ontology management infrastructure:

Integration of Heterogeneous Knowledge Sources: In general, answering questions as the above might however require uniform access to different sources or repositories. However, a common limitation is that different knowledge sources use different taxonomies and different metadata schemes to describe the stored content, which requires a user to switch between different representations depending on the backend system searched. A particular challenge is the integration of structured knowledge sources (e.g., document metadata) and unstructured sources (e.g., document fulltexts). We would like to access heterogeneous knowledge sources via a single, unified interface that integrates different metadata schemes as well as different topic hierarchies. In Sect. 2.3 we illustrate how the integration of heterogeneous knowledge sources can be realized using ontology mappings.

Automatic Content Extraction and Classification: Questions 3 and 4 might require fine-grained access to the topics of articles. Hereby, with topics we mean items of some classification scheme to which documents are related. Though in many cases articles are classified, we can not expect that all relevant categories are

actually explicitly stated. Thus, some support for automatically capturing the content of new documents added to the library seems necessary. Typically, the content of a knowledge source is not static, but changes over time: New documents come in, but also documents may be removed from the document base. We here require means to automatically extract relevant knowledge and to classify the new content. To address this requirement, we present in Sect. 2.4 techniques of incremental ontology learning.

Dealing with Inconsistent Information: In many cases, the information derived from diverse sources leads to inconsistencies. This is especially the case if information is derived using automatic knowledge acquisition tools such as wrappers, information extraction systems, or tools for automatic or semi-automatic ontology learning that aim at the semi- or even fully automatic extraction of ontologies from sources of textual data. Ontology-based applications which rely on learned ontologies have to face the challenge of reasoning with large amounts of inconsistent information resulting from automatic ontology generation systems. In Sect. 2.5 we introduce approaches that allow to process inconsistent ontologies.

Support for Structured Queries Against Metadata and Documents: With current interfaces to knowledge repositories, users either pose keyword-based queries on document fulltexts or abstracts or they use metadata search facilities to perform document retrieval. In general we are interested in receiving as answers facts from the knowledge base rather than only relevant documents.

We thus require the capability to pose structured queries to the underlying digital library which can be evaluated against the articles metadata as contained in the knowledge base. In general, we would like to allow the retrieval of facts besides the retrieval of documents. For metadata queries, current interfaces either offer preselected attribute fields (which are interpreted as conjunctive queries over a attribute-specific fulltext search) or they require some kind of formal query language.

In order to provide intuitive access and not to impose the burden on the user of learning a formal query language, we would like to allow the user to use an intuitive query language, ideally arbitrary natural language queries. In our running scenario, Bob should thus be able to ask the questions directly in the way they are stated in the scenario above. How such query answering can be realized, is discussed in Sect. 2.6.

Finally, we discuss Related Work in Sect. 2.7 and conclude with an outlook in Sect. 2.8.

2.2 Ontology Management Architecture

In the last decade, a number of competitive ontology management systems have been designed and implemented. Most naturally, many of these systems have a similar setup, i.e. they are built out of a similar set of basic components, each of which is responsible for a bundle of core functionalities. Despite these similarities, the systems also differ in many details such that ontology languages or query

languages supported. In this chapter, we do not intend to present a fixed architecture, but rather an architecture that contains the most typical components of ontology management systems. In some way, the architecture is inspired by existing ontology management systems such as KAON2[1] or OntoBroker.[2]

Figure 2.1 illustrates a typical ontology management system architecture. In the following, we shortly summarize the main components. The core of the overall architecture is the ontology model module. The ontology model, typically an in-memory representation, maintains a set of references to the ontologies, their respective contents and corresponding instance data sources. From a logical perspective, it can be seen as a set of (uninterpreted) axioms. The ontology model can typically be modified by means of an API or via well-defined interfaces. Ontology editors and ontology learning systems usually access and modify the contents of the ontology model. As a separate component, the ontology metadata store maintains metadata information about the ontologies themselves, for example author, version and compatibility information.

While the ontology model (or parts of it) is usually kept in-memory during runtime, the underlying layer is responsible for persistence and storage functionalities. Often, these functionalities will be internal serialization and import/export. However, it may also include connectors to external data sources, e.g. classical relational databases that are interpreted in the context of the ontology model they are embedded in.

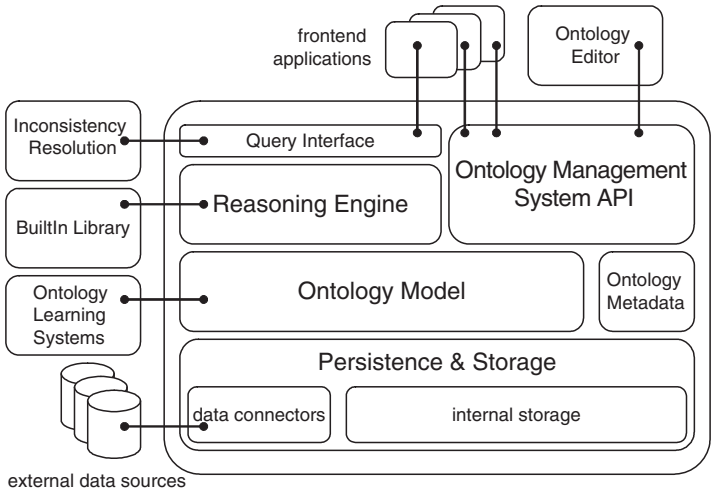

Fig. 2.1 Ontology management system architecture

[1] See http://kaon2.semanticweb.org/

[2] See http://www.ontoprise.de/content/e1171/e1231/index_eng.html

The reasoning engine operates on top of the ontology model, giving the set of axioms an interpretation and thus deducing new facts from the given primitives. Obviously, the behaviour of the reasoning engine will depend on the supported semantics, e.g. Description Logics in the case of OWL reasoning engines. The interaction with the reasoning engine from outside takes place either by means of a defined API or by means of a query interface. Again, the query interface needs to support a standardized query language, e.g. SPARQL for OWL/RDF data. The basic reasoning capabilities based on the ontology model can be further extended by means of external Builtins. Builtins can be seen as external, black-box type components that compute the extensions of certain properties externally. This mechanism increases the flexibility for knowledge representation by e.g. allowing complex arithmetic calculations, comparisons of arbitrary data types and so on. As a further component, the inconsistency resolution module can interact with the query engine and the reasoning engine to return meaningful answers to certain queries, even if the underlying knowledge base is inconsistent.

Based on this architecture, advanced front-end applications can be designed that interact with the ontology management system. As an example, consider the case of a natural language query interface that allows the translation of queries posed in everyday natural language into structured SPARQL queries that are in turn answered by the query interface.

2.3 Ontology Mapping

In large distributed information systems, the individual data sources are often described based on different, heterogeneous ontologies. To enable interoperability across these data sources, it is necessary to specify how the data residing at a particular node corresponds to data residing at another node. This is formally done using the notion of a mapping. There are mainly two lines of work connected to the problem of ontology mapping: (1) identifying correspondences between heterogeneous ontologies, (2) representing these correspondences in an appropriate mapping formalism and using the mappings for a given integration task. We here briefly discuss each of these individual aspects.

2.3.1 Mapping Discovery

The first task of identifying correspondences between heterogeneous ontologies is also referred to as mapping discovery or ontology matching. Today, there exist a number of tools that support the process of ontology matching in automated or semi-automated ways. A good survey of such ontology matchers is provided in (Shvaiko and Euzenat 2005). Instead of presenting individual tools, we here describe generic process for ontology matching, as first it was first presented for

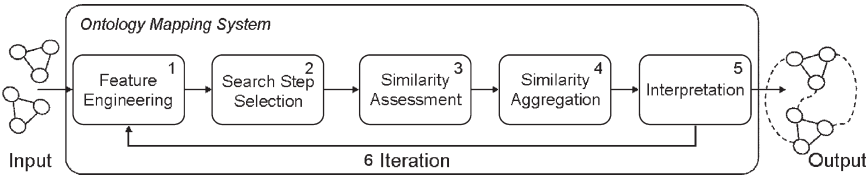

Fig. 2.2 Ontology matching process

the FOAM system in (Ehrig and Sure 2004). Other ontology matching approaches can be described in terms of this process as well. Figure 2.2 illustrates the six main steps of the generic process.

Input: Input for the process are two or more ontologies, which need to be matched with one another. Additionally, it is often possible to enter preknown (manual) matches. They can help to improve the search for matches.

1. **Feature Engineering:** The role of feature engineering is to select relevant features of the ontology to describe specific ontology entities, based on which the similarity with other entities will later be assessed. For instance, the matching process may only rely on a subset of OWL primitives, possibly only the taxonomy, or even just the linguistic descriptions of entities. For each feature, a specific matcher based on some similarity measure will be assigned.
2. **Search Step Selection:** The derivation of ontology matches takes place in a search space of candidate matches. This step may choose to compute the similarity of certain candidate element pairs and to ignore others in order to prune the search space.
3. **Similarity Computation:** For a given description of two entities from the candidate matches this step computes the similarity of the entities using the selected features and corresponding similarity measures.
4. **Similarity Aggregation:** In general, there may be several similarity values for a candidate pair of entities, e.g., one for the similarity of their labels and one for the similarity of their relationship to other entities. These different similarity values for one candidate pair have to be aggregated into a single aggregated similarity value. Often a weighted average is used for the aggregation of similarity values.
5. **Interpretation:** The interpretation finally uses individual or aggregated similarity values to derive matches between entities. The similarities need to be interpreted. Common mechanisms are to use thresholds or to combine structural and similarity criteria. In the end, a set of matches for the selected entity pairs is returned.
6. **Iteration:** The similarity of one entity pair influences the similarity of neighboring entity pairs, for example, if the instances are equal this affects the similarity of the concepts and vice versa. Therefore, the similarity is propagated through the ontologies by following the links in the ontology. Several algorithms perform an iteration over the whole process in order to bootstrap the amount of structural knowledge. In each iteration, the similarities of a candidate alignment are

recalculated based on the similarities of neighboring entity pairs. Iteration terminates when no new alignments are proposed.

Output: The output is a representation of matches and possibly with additional confidence values based on the similarity of the entities.

Current ontology matchers often represent mappings merely as a set of equivalences between simple ontology elements. While this is useful for elementary mappings, it is not sufficient for more advanced integration tasks involving ontology mappings, as we discuss in the following.

2.3.2 Mapping Representation

In contrast to the area of ontology languages where the Web Ontology Language OWL has become a de facto standard for representing and using ontologies, there is no agreement yet on the nature and the right formalism for defining mappings between ontologies. We here briefly discuss general aspects of formalisms for mapping representation.

What do mappings define? We can see mappings as axioms that define a semantic relation between elements in different ontologies. Most common are the following kinds of semantic relations:

Equivalence (≡): Equivalence states that the connected elements represent the same aspect of the real world according to some equivalence criteria. A very strong form of equivalence is equality, if the connected elements represent exactly the same object.

Containment (⊆): Containment states that the element in one ontology represents a more specific aspect of the world than the element in the other ontology. Depending on which of the elements is more specific, the containment relation is defined in the one or in the other direction.

Overlap (*o*): Overlap states that the connected elements represent different aspects of the world, but have an overlap in some respect. In particular, it states that some objects described by the element in the one ontology may also be described by the connected element in the other ontology.

In some approaches, these relations are supplemented by their negative counterparts. The corresponding relations can be used to describe that two elements are *not* equivalent, *not* contained in each other or *not* overlapping (or disjoint) respectively.

What do mappings connect? Most mapping languages allow defining the correspondences of mappings between ontology elements of the same type, e.g. classes to classes, properties to properties, etc. These mappings are often referred to as simple, or one-to-one mappings. While this already covers many of the existing mapping approaches, there are a number of proposals for mapping languages that rely on the idea of view-based mappings and use semantic relations between queries to connect models, which leads to a considerably increased expressiveness.

For example, the approach of (Haase and Motik 2005) allows describing correspondences between conjunctive queries over the source and target ontologies. The expressiveness of conjunctive queries corresponds to that of the well-known select-project-join queries in relational databases. Two typical approaches to specify mappings are the *global-as-view* (GAV) approach, where elements of the target are described in terms of queries over source, and the *local-as-view* (LAV) approach, where elements of the source are described in terms of queries over target.

Possible applications such mappings are manifold, including for example data transformation and data exchange. However, most commonly mappings are applied for the task of *ontology-based information integration*, which addresses the problem of integrating a set of local information sources, each described using a local ontology. Queries are expressed against a global, integrated ontology, which provides a unified view of the local ontologies. For details about query answering algorithms in such integration systems, we refer the reader to (Haase and Motik 2005).

2.4 Incremental Ontology Learning

Being an important means of data interchange and integration ontologies have become a solid basis for an increasing number of knowledge-based applications. Depending on the particular requirements their size ranges from a few to a few thousands of classes, and reasoning applications demand for more and more complex axiomatizations. Since building ontologies therefore is a task which is often difficult and very time consuming, it is desirable to automatize the construction of ontologies as far as possible.

Ontology learning aims at the automatic generation of ontologies from unstructured or semi-structured documents by machine learning, or natural language processing techniques. Whereas the learned ontologies mostly consist of concepts, instances, as well as taxonomic and non-taxonomic relationships recently a few approaches towards learning more expressive ontologies have been developed (e.g., Völker et al. 2007a). Common to all of these approaches is the need for timely adaptation of ontologies over time. That is, as the underlying data sources and the knowledge represented by them changes the learned ontologies have to be modified to reflect the new knowledge.

An efficient way of such data-driven ontology evolution is often referred to as incremental ontology learning. Frameworks like, for example, Text2Onto (Cimiano and Völker 2005) are able to suggest ontology updates based on changing lexical evidence and so-called document pointers, i.e. references linking individual ontology elements to fragments of text or other resources. Keeping document pointers allows for tracing results back to the data, but also enables applications to access parts of the document content by means of the ontology. Since a learned ontology can be considered as some kind of abstract, formal model of the knowledge contained in the underlying documents ontology learning can be a suitable means to support question answering tasks involving formal queries to unstructured, mostly textual data.

Obviously, as soon as more expressive constructs, in particular any type of negation (e.g., disjointness, Völker et al. 2007b), or cardinality constraints are introduced into learned ontologies the adaptation of ontologies over time may lead to logical inconsistencies. Similar problems arise if multiple ontologies are generated from related resources, and have to be integrated into one overall ontology by ontology mapping or merging techniques. In order to address these issues, future environments for automatic or semi-automatic ontology engineering will require methodologies for integrating ontology learning and reasoning, particularly including techniques for debugging or consistent ontology evolution (Haase and Völker 2005).

2.5 Processing of Inconsistent Ontologies

Dealing with inconsistencies is an important issue in the Semantic Web because of its scalability, distribution, and multi-authorship. The classical entailment in logics is explosive: any formula is a logical consequence of a contradiction. Therefore, conclusions drawn from an inconsistent ontology by classical inference may be completely meaningless.

There are two main ways to deal with inconsistent ontologies. One is to diagnose and repair inconsistencies. Another approach is to apply a non-standard reasoning method to obtain meaningful answers in the presence of inconsistency. The former is called the approach of *debugging* inconsistent ontologies, whereas the latter is called the approach of reasoning with inconsistent ontologies. In this section, we will overview both approaches of processing inconsistent ontologies. In particular, we will examine them by introducing the work has been done in the SEKT project.[3]

2.5.1 Debugging Inconsistent Ontologies

2.5.1.1 Framework and Tool

Debugging inconsistent ontologies is to diagnose and repair them. There exist several different approaches for debugging inconsistent ontology. One is to explain inconsistency by pinpointing. That can be done by a top-down approach with axiom pinpointing, or by a bottom-up approach by the support of an external reasoner. Another one is model-based diagnosis approach, which uses the traditional diagnosis approaches in the AI community (Reiter 1987).

Schlobach and Cornet (2003) proposes a non-standard reasoning service for debugging inconsistent terminologies. The work is further conducted in the SEKT

[3] http://www.sekt-project.com/

project (Schlobach and Huang 2007). In Description Logic, a concept is unsatisfiable if its interpretation is an empty set. An ontology is incoherent if it contains an unsatisfiable concept. Axiom pinpointing means identifying debugging-relevant axioms, where an axiom is relevant if an incoherent ontology becomes coherent once the axiom is removed or if, at least, a particular, previously unsatisfiable concept turns satisfiable. In order to obtain the pinpoints, Schlobach and Cornet introduce the notions of MIPS and MUPS. MIPS are incoherent sets of terminological axioms which can, possibly, not be traced back to the unsatisfiability of a particular concept name. If we are interested in the causes for unsatisfiability of a particular concept we need a more fine-grained notion, called minimal unsatisfiability-preserving sub-TBoxes (MUPS) of a TBox and a particular concept. In general there are several of these sub-TBoxes and we select the minimal ones, i.e., those containing only axioms that are necessary to preserve unsatisfiability.

DION is a system that uses the bottom-up approach for debugging inconsistent. DION uses an informed bottom-up algorithm to calculate MUPS/MIPS with the support of an external DL reasoner. The main advantage of the bottom-up approach is that it can deal with any DL-based ontology supported by an external reasoner. DION is designed to be a simple API for a general debugger with inconsistent ontologies. It supports extended DIG requests from other ontology applications, including OWL and DIG.[4] This means that DION can be used as an interface of an inconsistent ontology debugger as it supports the functionality of the underlying reasoner by just passing requests on and provides reasoning functionalities if needed. Therefore, the implementation of DION will be independent of those particular applications or systems. The prototypes of DION are available at the DION website.[5]

2.5.1.2 Experiment and Example

In the previous section of this chapter, we described the efforts of the Text2Onto team in learning expressive ontologies. Often the potential of ontologies cannot be fully exploited, because information such as disjointness of concepts is not made explicit, and as a consequence, possible modeling errors remain undetected. There are different methods to estimate disjointness of two concepts, e.g. based on taxonomic overlap, semantic similarity, linguistic patterns etc., which can be automatically combined by a machine learning classifier. Given a gold standard, this classifier would be trained to predict whether an unseen pair of two concepts is disjoint or not. Obviously, the problem of this approach is that a gold standard is needed, i.e. an agreed-upon collection of pairs of disjoint and non-disjoint concepts. Several experiments to construct such a gold standard have been conducted.

[4] http://dl.kr.org/dig/

[5] http://wasp.cs.vu.nl/sekt/dion

In these experiments, a sub-ontology PROTON is selected. Each concept pair is randomly assigned to six different people – three from each of two groups: the first one consisting of ontology experts, the second is composed of undergraduate students without profound knowledge in ontological engineering. Each of the annotators was given between 385 and 406 pairs along with natural language descriptions of the classes whenever those were available. Furthermore, they computed the majority votes for all the mentioned above datasets by considering the individual taggings for each pair. Adding the created disjointness statements to PROTON results in inconsistent PROTON ontologies. For example, there are 24 unsatisfiable concepts in the PROTON ontology with the disjointness created by 50% votes by Students.

Some of the most important findings from these experiments are that disjointness seems to be very difficult to understand for humans, and that it can be captured surprisingly well by automatic means. More surprising is that the agreement among the experts was not much higher than the agreement among the students. Another finding of the experiments was that even disjointness axioms introduced by full agreement of all annotators can, if included in the ontology, lead to incoherence, i.e. logical contradictions. To investigate the cause of this contradiction we applied DION to debug the newly constructed ontologies.

For the unsatisfiable concepts we calculate the minimal subsets of the ontology (MUPS) in which the concept is still unsatisfiable. Basically, this means that these sub-ontologies still contain a logical contradiction. It is interesting to observe that 100% of experts and students agree with the following axioms, however, they are incoherent:

{Reservoir → Lake, Lake → WaterRegion, Reservior → HydrographicStructure, HydrographicStrure → Facility, Disjoint(WaterRegion, Facility)}

The conclusion Reservoir → WaterRegion follows from the first two axioms, whereas Reservoir → Facility follows from the third and the fourth axioms. However, it contradicts with the axiom that WaterRegion and Facility are disjoint. This case shows that inconsistency occurs much more easily than people expect.

From these MUPS we calculate MIPS. These MIPS contain at least one “error”, they are orthogonal to classical “diagnoses”. They are not unique. We observe that the ontology including PROTON and only disjointness axioms agreed by all annotators only contains three unsatisfiable concepts. The situation is much worse in the case when disjointness axioms are added to PROTON, which were agreed upon by three experts. In this case, an analysis using a debugging tool such as DION can even be more useful to determine the cause of a logical error. Moreover, confidence or relevance information acquired during the ontology learning process might help to select the axiom, which is most likely to be incorrect.

2.5.2 Reasoning with Inconsistent Ontologies

2.5.2.1 Framework and Tool

Reasoning with inconsistent ontologies is to simply avoid the inconsistency by applying a non-standard reasoning method to obtain meaningful answers.

This approach is more suitable for the setting in the web area. For example, in a typical Semantic Web setting, one would be importing ontologies from other sources, making it impossible to repair them, and the scale of the combined ontologies may be too large to make repair effective.

The general task of a system of reasoning with inconsistent ontologies is: given an inconsistent ontology, return *meaningful* answers to queries. Huang et al. (2005) develops a general framework of reasoning with inconsistent ontologies, in which relevance based selection functions are used to obtain meaningful answers, where meaningfulness means that the answer is supported by a selected consistent sub-ontology of the inconsistent ontology. The main idea of the framework is: given a selection function, which can be defined as either syntactic or semantic relevance, we select some consistent sub-theory from an inconsistent ontology. Then we apply standard reasoning on the selected sub-theory to find meaningful answers. If a satisfying answer cannot be found, the relevance degree of the selection function is made less restrictive thereby extending the consistent sub-theory for further reasoning.

A linear extension strategy is proposed for selecting consistent sub-ontologies. We have proved that a reasoner using a linear extension strategy is never over-determined, may be undetermined, is always sound, and is always meaningful (Huang et al. 2005). A reasoner using a linear extension strategy is useful to create meaningful and sound answers to queries. The disadvantage of the linear strategy is that it may lead to an inconsistency reasoner that is undetermined. There exist other strategies that can improve the linear extension approach, for example, by backtracking and heuristics evaluation.

Selection functions play the main role in the framework of reasoning with inconsistent ontologies. Such a selection function determines which consistent subsets of an inconsistent ontology should be considered in its reasoning process. The general framework is independent of the particular choice of selection function. The selection function can either be based on a syntactic approach, or based on semantic relevance like for example in computational linguistics as in Wordnet or based on semantic relevance which is measured by the co-occurrence of concepts in the Web, which information is provided by search engines like Google.

PION is a system of reasoning with inconsistent ontologies, which is developed based on this framework. PION provides various selection functions and extension strategies to deal with inconsistent ontologies. PION supports syntactic-relevance-based selection functions and Google-based semantic relevance selection functions. The prototype of PION is available at its website.[6]

2.5.2.2 Experiment and Example

Huang and van Harmelen (2006) reports several experiments of PION. The tests show that the syntactic relevance approach works well with real-world inconsistent

[6] http://wasp.cs.vu.nl/sekt/pion

ontologies. An explanation for this could be that the syntactic relevance approach mimics our intuition that real-world truth is generally preserved best by the argument with the shortest number of steps; and whatever process our intuitive reasoning uses, it is very likely that it would somehow privilege just these shortest path arguments.

For example, the syntactic relevance approach works very well for the penguin example in which *Birds* are specified as *FlyingAnimals* and *Penguins* are specified as *Birds* which cannot fly. In this example, the reasoning path from *Penguin* to *Not Fly* is shorter than that from *Penguin* to *Fly*.

However, it does not work very well on the MadCow example, in which *Cows* are specified as *Vegetarians* whereas *MadCows* are specified as *Cows* which eats brains of sheep. Under the syntactic relevance based selection function, the reasoner returns that the "accepted" answer to the query "is *MadCow* a *Vegetarian*?" This counter-intuitive answer results from the main feature of the syntactic relevance approach, because it always prefers a shorter relevance path when a conflict occurs. In the MadCow example, the path "mad cow-cow-vegetarian" is shorter than the path "mad cow-eat brain-eat bodypart-sheep are animals-eat animal-NOT vegetarian". Therefore, the syntactic relevance-based selection function finds a consistent sub-theory by simply ignoring the fact "sheep are animals". The problem results from the unbalanced specification between *Cow* and *MadCow*, in which *Cow* is directly specified as a vegetarian whereas there is no direct statement "a *MadCow* is not a *Vegetarian*".

The semantic relevance approach provides an alternative to solve the limitation of the syntactic relevance approach. Cilibrasi and Vitanyi (2007) introduce the notion of Normalized Google Distances (NGD) to measure the co-occurrence of two keywords over the Web. NGD is introduced in PION as a measurement of semantic relevance between two concepts for its selection functions. For example, in the MadCow ontology, the Google-based semantic distance between *MadCow* and *Sheep* is shorter than that between *MadCow* and *Grass*, because *NGD(MadCow, Sheep) = 0.621* and *NGD(MadCow, Grass) = 0.723*. Namely, eating brains of sheep is considered as a more distinctive feature of MadCow than eating grass, based on the semantic information over the Web. Therefore, it provides an alternative to avoid the counter-intuitive answer if *MadCow* is specified as a *Cow* which eats sheep whereas *Cow* is specified as an animal which eats grass by default in the MadCow ontology. Huang and van Harmelen (2006) report several experiments of PION with various selection functions. The tests show that the semantic relevance based selection function plays better than the syntactic relevance selection function in reasoning with inconsistent ontologies.

2.6 Query Answering

Query answering is an important tool in ontology management. We here consider queries are logical conjunctive queries against the ontology. Query answering amounts to a reasoning process over the knowledge sources according to the

semantics of the underlying ontology language OWL. A query language of particular importance is SPARQL, which is currently being standardized by the W3C as the standard query language for the Semantic Web. SPARQL was originally designed as RDF query language. As every OWL ontology can be encoded in an RDF graph, SPARQL can serve – at least syntactically – as an OWL query language.

The SPARQL query language is based on matching graph patterns. The simplest graph pattern is the triple pattern. Triple patterns in queries additionally may contain variables in the subject, predicate or object positions. Combining triples gives a basic graph pattern, where an exact match to a graph is needed to fulfil a pattern.

Queries are composed of two main building blocks, the SELECT or CONSTRUCT clause and the WHERE clause. The SELECT or CONSTRUCT clause identifies the variables to appear in the query results, they thus correspond to the distinguished variables of a conjunctive query. The WHERE clause identifies the triple pattern to match. As an example, consider the following SPARQL query which asks for articles that are associated with the topic "Intellectual Capital"?:

```
SELECT ?x WHERE {
?x rdf:type Article.
?x hasSubject ?y.
?y rdfs:label ?z.
match(?z, "Intellectual Capital")
}
```

For the answering of queries, we follow a virtual integration approach that does not require a full integration of all knowledge sources: The actual data still resides in the individual knowledge sources, and the mapping between the ontology and the knowledge sources is only used at runtime to retrieve the facts that are actually relevant for answering a query. In a query, predicates can be used that require access to different knowledge sources. The evaluation of these predicates is automatically *pushed down* to the respective knowledge sources. For example, this could be a relational query in the case of a relational database, or to a fulltext index in the case of the fulltext match predicate *"match"* over the topic hierarchy contained in the above query whereby the latter is implemented as a so-called *built-in*.

2.6.1 Extending Ontologies with Built-ins

Ins many situations it is desirable to let the reasoning engine work with knowledge that can be deduced from the available knowledge by using external components without relying on the reasoning within the semantics of the underlying logic. Built-ins are an approach to address this requirement. Built-ins can be seen as special purpose predicates defined on individuals or datatypes within the ontology. In contrast to conventional predicates, the extension of these predicates, each of which is associated with a particular software component, is not maintained within the ontology model but is computed on-the-fly during query evaluation.

Consider the following examples:

- A predicate *"stringmatch"* defined on string data types which computes the matching between two strings according to some notion of "match" (e.g., whether the strings differ only in their capitalization).
- A predicate *"classify"* defined on a string and an individual that returns the result of a text classifier against a previously trained category with the corresponding model being stored within the second argument.
- A predicate *"similarity"* that returns the similarity between two argument individuals according to a specified similarity measure.[7]

2.6.2 Expressive Natural Language Interfaces

Certainly, conjunctive queries in a query language like SPARQL cannot expected to be expressed directly by an end user. To address this problem, different approaches have been proposed to automatically translate queries in formalisms conceptually more adequate for end users (e.g., keyword queries, or queries in natural language) into formal queries based on logical languages. For example, ORAKEL (Cimiano et al. 2007) is a natural language interface which translates natural language queries to structured queries formulated with respect to a given ontology. This translation relies essentially on a compositional semantic interpretation of the question guided by two lexica: a domain-specific and a domain-independent lexicon. As the name suggests, the domain-independent lexicon is a priori encoded into the system and captures the domain-independent meaning of prepositions, determiners, question pronouns etc., which typically remain constant across domains. The domain-specific lexicon needs to be created by a *lexicon engineer* which is assumed to create the lexicon in various cycles. The end users then are able to directly interact with the application via natural language questions, which are translated into SPARQL queries by the ORAKEL system. The underlying mechanism however is hidden from the users.

2.7 Discussion and Related Work

Ontology management technologies are naturally connected to a wide variety of other areas in the field of semantically enabled knowledge management technologies. In this section, we shortly sketch relevant connections to two prominent areas which have been exploited as part of the SEKT project: knowledge discovery and natural language processing.

[7] E.g. consider the KAON2 similarity project at http://ontoware.org/projects/kaon2similarity/

2.7.1 Relation to Knowledge Discovery

The relation between ontology management and knowledge discovery is obviously bi-directional. On the one hand, knowledge discovery techniques, in particular text mining, can be exploited to generate ontological structures. On the other hand, ontologies can support the knowledge discovery process. For the first case, a recent approach in the SEKT project has used techniques for the scalable discovery of topic ontologies from text documents based on Latent Semantic Indexing and KMeans clustering) (c.f., Fortuna et al. 2006a, 2006b). The resulting system, OntoGen, offers supports to the user during the construction process by suggesting topics and analyzing them in real time. Following a somewhat different paradigm, machine learning techniques can be used to process data contained in the ontology, as in the above example of a text classification built-in. From the other perspective, the structure of ontologies has been used to exploit the implicit similarity of the concept nodes in text mining tasks (c.f., Bloehdorn et al. 2006).

2.7.2 Relations to Natural Language Processing (NLP)

As ontologies formally reflect conceptualizations of domain knowledge, there is a natural relation to applications that deal with the processing of natural language, which can be seen as the most flexible, though informal knowledge representation paradigm. The connections are present in both directions. One the one hand, applications that deal with natural language can rely on ontological knowledge structures to process unstructured language input. The ontologies can contain valuable background information that can help to interpret the input, allow for sanity checks of the results. Further the ontologies can provide the formal framework to represent the results of the NLP analysis as for example in the case of information extraction (IE), where the extracted entities are typically annotated against an existing ontology. On the other hand, NLP techniques can enable new functionalities for the ontology management system, e.g. allow for natural language queries to the ontology and knowledge base as in the above example of the ORAKEL system. Most naturally, ontology learning techniques (Sect. 2.4), that deal with the transformation of textual input into formal knowledge structures rely on NLP techniques.

2.7.3 Conclusion and Outlook

We have presented the main building blocks of an ontology management system. Guided by the scenario of information integration in a digital library scenario, we have looked at the overall architecture, mapping, ontology learning, built-ins and query answering functionalities. Instantiations of the named components have been

developed within the EU integrated project "Semantic Knowledge Technologies (SEKT)". The guiding scenario has been prototypically implemented within one of the case studies in the SEKT project, i.e. the BT digital library – a detailed description of which is given in Chap. 14.

The results of SEKT and in particular the results on ontology management are further developed in various projects. The EU integrated project NeOn[8] aims at extending the existing infrastructure such that one is able to deal with networked ontologies, i.e. multiple, distributed but interconnected ontologies. Further issues addressed by NeOn include managing the dynamics and evolution of ontologies and providing support for collaborative development of networked ontologies. As another example, the EU integrated project XMedia[9] addresses the issue of knowledge management with ontology management systems in complex distributed environments. A particular focus of the project are effective and efficient paradigms for knowledge retrieval within ontologies which allow users to define and parameterize views on the available knowledge according to their needs. Further aspects of the project are techniques for knowledge fusion and uncertainty representation and handling within ontology management systems.

In summary, integrated ontology management systems provide a flexible means for integrating and querying diverse and heterogeneous knowledge sources. The ontology management system can flexibly built upon existing data sources and connect to other applications but – as such – also forms the basis for a wide range of semantically enabled applications.

Acknowledgement This work was supported by the IST Programme of the European Community under SEKT (IST-IP-2003-506826).

References

Bloehdorn S, Basili R, Cammisa M and Moschitti A (2006). Semantic Kernels for Text Classification Based on Topological Measures of Feature Similarity. Proceedings of the 6th IEEE International Conference on Data Mining (ICDM'06) (Hong Kong, December 18–22, 2006).

Cilibrasi R and Vitanyi P (2007). The Google similarity distance. IEEE/ACM Transactions on Knowledge and Data Engineering, 2007.

Cimiano P and Völker J (2005). Text2Onto – A Framework for Ontology Learning and Data-Driven Change Discovery. Proceedings of the 10th International Conference on Applications of Natural Language to Information Systems (NLDB), 2005.

Cimiano P, Haase P and Heizmann J (2007). Porting Natural Language Interfaces Between Domains: An Experimental User Study with the ORAKEL System. Proceedings of the 12th International Conference on Intelligent User Interfaces (Honolulu, Hawaii, USA, January 28 – 31, 2007). IUI'07. ACM Press, New York, 180–189.

Ehrig M and Sure Y (2004). Ontology Mapping – An Integrated Approach. Proceedings of the First European Semantic Web Symposium, Heraklion, Greece, 76–91.

[8] http://www.neon-project.org

[9] http://www.x-media-project.org

Fortuna B, Grobelnik M and Mladenić D (2006a). System for Semi-automatic Ontology Construction, Demo at ESWC 2006, June 11–14, 2006, Budva, Montenegro.

Fortuna B, Grobelnik M and Mladenić D (2006b). Background Knowledge for Ontology Construction, Poster at WWW 2006, May 23–26, 2006, Edinburgh, Scotland.

Haase P and Motik B (2005). A Mapping System for the Integration of OWL-DL Ontologies. Proceedings of the First International Workshop on Interoperability of Heterogeneous Information Systems (Bremen, Germany, November 4, 2005). IHIS'05. ACM Press, New York, 9–16.

Haase P and Völker J (2005). Ontology Learning and Reasoning – Dealing with Uncertainty and Inconsistency. Proceedings of the Workshop on Uncertainty Reasoning for the Semantic Web (URSW), 2005.

Huang Z and van Harmelen F (2006). Reasoning with Inconsistent Ontologies: Evaluation, SEKT Deliverable D3.4.2.

Huang Z, van Harmelen F and ten Teije A. (2005). Reasoning with Inconsistent Ontologies. Proceedings of IJCAI'05.

Reiter R (1987). A theory of diagnosis from first principles. Artificial Intelligence, 32(1):57–95.

Schlobach S and Cornet R (2003). Non-Standard Reasoning Services for the Debugging of Description Logic Terminologies. Proceedings of IJCAI'03, Morgan Kaufmann, San Fransisco.

Schlobach S and Huang Z (2007). Inconsistent Ontology Diagnosis and Repair, SEKT Deliverable D3.6.3.

Shvaiko P and Euzenat J (2005). A survey of schema-based matching approaches. Journal on Data Semantics IV, 4 (Lecture Notes in Computer Science, vol. 3730):146–171. Springer, Berlin Heidelberg.

Völker J, Hitzler P and Cimiano P (2007a). Acquisition of OWL DL Axioms from Lexical Resources. Proceedings of the 4th European Semantic Web Conference (ESWC'07) (Lecture Notes in Computer Science), vol. 4519. Springer, Berlin Heidelberg.

Völker J, Vrandečić D, Sure Y and Hotho A (2007b). Learning Disjointness. Proceedings of the 4th European Semantic Web Conference (ESWC) (Lecture Notes in Computer Science), vol. 4519. Springer, Berlin Heidelberg.