Chapter 5: Ontology management - Storing, aligning and maintaining ontologies

Michel Klein, Ying Ding, Dieter Fensel, Borys Omelayenko

1 Ontologies need to be managed

In the OnToKnowledge project, several ontologies are developed and used in a number of applications and case-studies. Those ontologies need to be stored, sometimes aligned and their evolution needs to be managed. All these tasks together are called *ontology management*. These issues are not only relevant in the OnToKnowledge project, but are also in a broader context.

Alignment is a central task in ontology reuse. Reuse of existing ontologies is often not requires considerable effort [Uschold et al, 1998]: the ontologies either need to be *integrated* [Pinto et al, 1999], which means that they are merged into one new ontology, or the ontologies can be kept separate. In both cases, the ontologies have to be aligned, which means that they have to be brought into mutual agreement. The problems that underlie the difficulties in integrating and aligning are the *mismatches* that may exist between separate ontologies. Ontologies can differ at the language level, which can mean that they are represented in a different syntax, or that the expressiveness of the ontology language is dissimilar. Ontologies also can have mismatches at the model level, for example in the paradigm, or modelling style [Klein, 2001].

Ontology alignment is also very relevant in a Semantic Web context. The Semantic Web will provide us with a lot of freely accessible domain specific ontologies. To form a real *web of semantics* – which will allow computers to combine and infer implicit knowledge – those separate ontologies should be aligned and linked.

Support for evolving ontologies is required in almost all situations where ontologies are used in real-world applications. In those cases, ontologies are often developed by several persons and will continue to evolve over time, because of changes in the real-world, adaptations to different tasks, or alignments to other ontologies. To prevent that such changes will invalidate existing usage, a change management methodology is needed. This involves advanced versioning methods for the development and the maintenance of ontologies, but also configuration management, that takes care of the identification, relations and interpretation of ontology versions.

All these aspects come together in integrated **ontology library systems**. When the number of different ontologies is increasing, the task of storing, maintaining and re-organising them to secure the successful re-use of ontologies is challenging. Ontology library systems can help in the grouping and re-organising ontologies for further re-use, integration, maintenance, mapping and versioning. Basically, a library system offers various functions for managing, adapting and standardising groups of ontologies. Such integrated systems are a required for the Semantic Web to grow further and scale up.

In this chapter, we will describe the results of the OnToKnowledge project with respect to the above mentioned areas. We start with a description of the alignment task and show a meta-ontology that is developed to specify the mappings. Then, we discuss the problems that are caused by evolving ontologies and describe two important elements of a change management methodology. Finally, in section 4, we

survey existing library systems and formulate a wish-list of features of an ontology library system.

2 Aligning ontologies

For effective ontology interoperation, ontologies must be efficiently aligned. These alignments must explicitly represent the maximal possible share of the relationships between the ontologies and their elements to enable efficient ontology reuse.

2.1 Why is aligning needed

The knowledge management scenario, which is in focus in the OnToKnowledge project, assumes different departments and individual employees to create domain-specific ontologies capturing specific aspects of their knowledge. Special mapping ontologies must be created to link different terminologies and modelling styles used in these domain specific ontologies, creating bridges between separated pieces of knowledge. These bridges along with domain ontologies are then used to perform cross-ontology information search and retrieval.

In the B2B area different process, document, and vocabulary ontologies are created by different companies instead of the domain ontologies created in the knowledge management tasks. Process ontologies capture the ordering of procurement events, types of attached documents, and inter-dependency of the documents. Document ontologies specify conceptual models of the documents together with constraints. Vocabularies contain hierarchies of terms used in the documents. The mapping ontologies specify the correspondence between different processes, documents and vocabularies.

Existing ontology mapping techniques primarily concern with weak ontology coupling [Mitra et al, 2000] needed to refer them and to query. In the business integration tasks mapping ontologies specify the transformations of instance documents and hence represent strong and well-grained correspondences. In this subsection we present an outline of a mapping meta-ontology that specifies a template for the B2B integration mapping ontologies.

2.2 The B2B mapping tasks

The document integration task [Omelayenko & Fensel, 2001], the kernel part of the B2B integration scenario, envisages the following document transformation chain. The transformation is performed between two XML documents of the source XML format and the target format. However, the complexity of the task does not allow to perform direct document transformation with XSLT [Clark, 1999] because the necessity to program numerous constraints and transformations for each new document format requires tremendous programming effort. Because of this the integration is performed via a mediating ontology that contains conceptual models for all the documents that appear during the integration process and necessary constraints to be checked during the transformation.

2.2.1 The transformation process

The transformation chain of a source document to the target document consists of the following steps as illustrated in Figure 1:

- The source XML document is transformed to its conceptual model in RDF that captures all the objects (with a possible small hierarchy of their classes) and relations presented in the document.
- The source conceptual model is aligned to the mediating ontology. These alignments allow applying the constraints presented in the mediating ontology to the source conceptual model. The document is then stored as an instance of the mediating ontology.
- The target RDF conceptual model corresponds to the target document and is aligned to the mediating ontology. The target document is extracted from the mediating document according to these alignments.
- The target XML document is re-constructed from its conceptual document.

This chain represents a light version of ontology aligning: different conceptual models are aligned to a mediating ontology. This requires aligning classes and properties, but does not include axiom mapping.

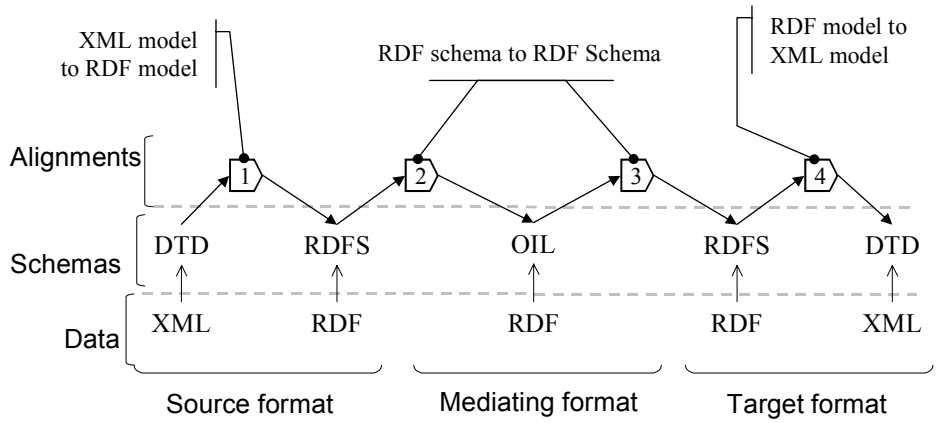

Figure 1. The alignments for the B2B document integration task

2.2.2 Mapping Meta-Ontology

We developed the RDFT (RDF Transformation) mapping meta-ontology[1] that specifies a small ontology for mapping XML DTDs to/and RDF Schemas and is built on top of RDF Schema. The basic class diagram is presented in Figure 2, where the classes are represented by their names, and name nesting indicates the is-a relationship. The main concept of RDFT is the bridge between two sets of concepts: the source set and the target set. The bridges are grouped into maps. Each **Map** is a collection of bridges serving a single purpose. The maps are identified by their names (URL's) and form minimal reusable module of RDFT bridges.

An abstract class **Bridge** describes common properties of bridges allowing only one-to-many and many-to-one bridges. Each Bridge contains the ValueCorrespondance property linking a map between the instance values of the source and target entities.

The bridges also contain the Relation property linking to one of the BridgeRelations: EquivalenceRelation or VersionRelation:

[1] http://www.cs.vu.nl/~borys/RDFT

- Equivalence bridges specify that the source element of a one-to-many bridge is equivalent to the target set of elements, and the source set of elements is equivalent to the target element for many-to-one bridges.
- A Version bridge specifies that the target set of elements form a (later) version of the source set of elements. Opposite to equivalence bridges, they assume that both source and target concepts belong to the same domain (or document standard), and may refer to two concepts with the same name (but different namespaces indicating versions), and imply that all the relations that held for the original concept must hold for the versioned concept, if the opposite is not stated explicitly.

Several types of Bridges are defined in RDFT:

- Class2Class and Property2Property bridges between RDF Schema classes and properties. In RDF Schema classes are represented by their names, place in taxonomy, and properties that are attached to this class. Properties are defined as first-class objects together with classes, and they capture most of domain knowledge [Lassila & Swick, 1999]. Classes specify aggregation of properties, and thus we do not include class-to-property and property-to-class bridges in RDFT. These bridges occur at steps 2 and 3 of the integration process depicted in Figure 1. Class2Class bridges between a set of *n* source classes and a set of *m* target classes declares that a set of *n* instances of the classes listed as sources corresponds to a set of *m* instances of the classes listed as targets. Property2Property bridges have similar semantics related to properties.

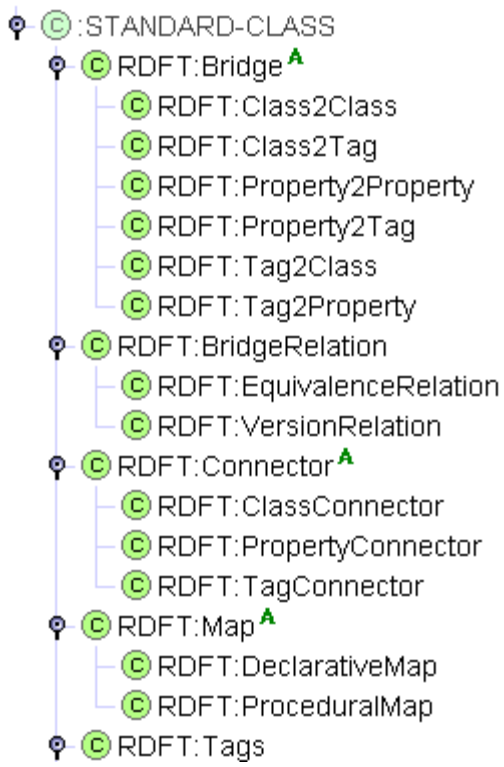

Figure 2. RDFT class diagram

- Tag2Class and Tag2Property bridges between XML tags of the source DTD and the target RDF Schema classes and properties. They occur at step 1 of the integration process.
- Class2Tag and Property2Tag bridges between RDF Schema classes and properties, and the elements of the target DTD. They occur at step 4 of the process.

All of the bridges contain the ValueCorrespondance property inherited from the abstract Bridge class linking to a Map. Two types of Maps are defined in RDFT:

- DeclarativeMap that specifies a set of bridges mapping all possible values to be mapped.
- ProceduralMap specifies an XPath [Clark, 1999] expression transforming instance data. XPath defines the means for two tasks: addressing data elements in XML documents and performing element or attribute value transformations (Chapter 4

of the specification[2]). In procedural maps we use only the second part of the XPath functions (e.g. substring_before).

The bridges are linked to the maps with different Connectors.

More information on RDFT is available from the RDF project homepage[3].

2.2.3 Mapping in OIL

The RDFT meta-ontology is intended to serve as a template for creation mapping ontologies for the business integration tasks. It represents meta-classes that are then instantiated into user's classes linking user's domain ontologies. OIL was developed as an ontology representation language and contains less flexible means for defining meta-classes as RDF Schema does. However, it is still possible to represent RDFT semantics in OIL.

We model the RDFT bridges in OIL in the following way. For each bridge we specify two classes that define the source and the target concepts to be mapped. For example, a one-to-many Class2Class bridge specifies the fact that each instance of the source class is equivalent to the set of instances, one instance of each target class. The correspondent bridge sources and bridge targets can be defined as follows:

```
class-def BridgeSources
  slot-constraint rdft_set_member has-value SourceClass

class-def BridgeTargets
  slot-constraint rdft_set_member has-value TargetClass1
  slot-constraint rdft_set_member has-value TargetClass2
  slot-constraint rdft_set_member has-value TargetClass3
```

The equivalence of BridgeSources to BridgeTargets can be also modelled in OIL by an equivalent axiom:

```
equivalent BridgeSources BridgeTargets
```

This axiom specifies the fact that each instance of the SourceClass class is equivalent to three instances, one of the TargetClass1, the second is of the TargetClass2, and the third is of TargetClass3.

However, statements over the bridges (e.g. Maps) need to be specified as statements over OIL axioms, and it is problematic to represent them in OIL directly. Despite of that bridge translation to OIL is still very useful because it allows to invoke an inference engine to perform knowledge-level validation of the bridges.

2.3 Evaluation of OTK techniques

Several techniques have been developed in the On-to-knowledge project that may be used in the B2B integration tasks. First, OIL can be used to model RDFT bridges and FaCT reasoner can be then used to check their consistency. Second, steps 2 and 3 of the transformation process depicted in Figure 1, deal only with RDF documents and their RDF Schemas. This allows using Sesame for storing and querying them.

---

[2] http://www.w3.org/TR/xpath

[3] http://www.cs.vu.nl/~borys/RDFT

3 Supporting ontology change

3.1 Ontologies are changing

Ontologies are a key technology in the OnToKnowledge project. All approaches for knowledge management and information retrieval in the project are based on such formal descriptions of particular domains. However, in practice those pieces of knowledge are not static, but evolve over time. Support to handle this evolution is needed. This is especially important when ontologies will be used in a decentralised and uncontrolled environment like the web, where changes occur without co-ordination. Much more than in a controlled environment, this may have unexpected and unknown results.

There are several reasons for changes in ontologies. According to Gruber [Gruber, 1993], an ontology is a *specification of a conceptualisation of a domain*. Hence, changes in ontologies can be caused by either:

- changes in the domain;
- changes in the conceptualisation;
- changes in the specification.

The first type of change is often occurring. This problem is very well known from the area of database schema versioning. In [Ventrone et al, 1991], seven different situations are sketched in which changes in a domain (domain evolution) require changes to a database model. An example of this type of change is the merge of two university departments: this is a change in the real world, which requires that an ontology that describes this domain is modified, too.

Changes in the conceptualisation are also frequently happening. It is important to realise that a *shared* conceptualisation of a domain – which is a requirement for information exchange – is not a static specification that is produced once in the history, but has to be reached over time. In Chapter 4, ontologies are described as dynamic networks of meaning, in which consensus is achieved in a social process of exchanging information and meaning. This view attributes a dual role to ontologies in information exchange: they provide consensus that is both a *pre-requisite* for information exchange and a *result* of this exchange process.

An conceptualisation can also change because of the usage perspective. Different tasks may imply different views on the domain and consequently a different conceptualisation. When an ontology is adapted for a new task or a new domain, the modifications represent changes to the conceptualisation. For example, consider an ontology about traffic connections in Amsterdam, with concepts like roads, cycle-tracks, canals, bridges and so on. When the ontology is adapted from a bicycle perspective to a water transport perspective, the conceptualisation of a bridge changes from a remedy for crossing a canal to a time consuming obstacle.

Finally, a specification change is a kind of translation, i.e., a change in the way in which a conceptualisation is formally recorded. Although ontology translation is an important and non-trivial issue in many practical applications, it is less interesting *from a change management perspective*, for two reasons. First, an important goal of a

translation is to retain the semantics, i.e., specification variants should be equivalent[4] and they thus only cause syntactic interoperability problems. Second, a translation is often created to use the ontology in an other context (i.e., an other application or system), which heavily reduces the importance of interoperability questions.

Changes in ontologies are thus inevitable. In the next sections, we will look at characteristics of ontology changes and describe two elements of an ontology change

3.2 Changes in ontologies involve several problems

There are several problems involved with ontology changes. In this section we will look at incompatibilities caused by ontology changes, the specification of them, and at the conceptual implication of such changes.

3.2.1 Effects of ontology change

An important effect of the evolution of ontologies is that it might cause incompatibilities. Incompatibility for ontologies means that the original ontology can not be replaced by the changed version without causing side effects in the conforming data or the applications that use them. However, the real problem is that these side effects, and thus the meaning of compatibility, depend on the *use* of the ontology.

- When an ontology is used to specify the meaning of data, this data may get an different interpretation or may use unknown terms. An example of this use is a web page which content is annotated with terms from an ontology.
- If ontologies are built from other ontologies, changes to the source ontology may affect the meaning of the resulting ontologies.
- Applications that use the ontology may also be hampered by changes to the ontology. In the ideal case, the conceptual knowledge that is necessary for an application should be merely specified in the ontology; however, in practice applications also use an internal model. This internal model may become incompatible with the ontology.

The meaning of compatibility is different for each of those types of usage. In the first case, compatibility means the ability to interpret all the data correctly through the changed ontology. This is much like the interpretation of compatibility in database schema versioning. Compatibility here means “preservation of instance data”.

In the second case, the effects of the changes on the logical model that the ontology forms are often important. Other ontologies that import an ontology might depend on the conclusions that can be drawn from the it. A change in the ontology should not make previous conclusions invalid. In this case, compatibility means “consequence preservation”.

Applications that use the ontology might depend on the logical model, but also on the characteristics of the ontology itself. For example, a web site that use an ontology for navigation can depend on the fact that there are only four top-level classes, or that the hierarchy is only three levels deep. A change that does not invalidate queries to

---

[4] Although in practice a translation often implies a change in semantics, possibly caused by differences in the representation languages. See for a exploration of ontology language differences and mismatches [Corcho et al, 2000] and [Klein, 2001].

instance data or the logical model might invalidate queries to the ontology itself. This interpretation of compatibility is "preservation of answers to ontology queries".

3.2.2 Typical changes and their specification

The specification of changes is another problem. There are many possible types of changes in ontologies, ranging from simple renamings to compound transformations. The specification of especially the latter is important, because the effect of a compound change can be different from the accumulated effect of steps that build the complex change [Lerner, 2000].

To make this more concrete, we will consider changes in a particular content standard, i.e. UNSPSC[5]. Content standards specify a standard hierarchy of products and services which can be used by companies to classify their actual products. This hierarchy can be considered as a simple ontology that specifies a consensus on the products that exist. Different companies that use the same content standard can easily communicate with respect to their products. Besides UNSPSC, which addresses a general and broad domain of products and services, there are several other standard classifications in use, e.g., RosettaNet[6], which is targeted at IT industry, and e@Class[7], another broad standard that originates from Germany.

These standards tend to change very often. For example, when we take a look at UNSPSC, we see the following:

- there were 16 updates between 31 January 2001 and 14 September 2001;
- each update contained between 50 and 600 changes;
- in 7,5 month, more than 20% of the current standard is changed!

Although some parts of the UNSPSC schema might be more stable than other parts, it is clear that this amount of changes cannot be ignored. Such a high change rate can quickly invalidate a lot of the actual classifications of products. For example, the product "Binding elements" in version 8.0 is removed from the standard and three new products are added in version 8.1 ("Binding spines or snaps", "Binding coils or wire loops", and "Binding combs or strips"). This means that all products that were classified as "Binding elements" are unclassified under the new version.

An analysis of differences between two version of content standards has yielded the following list of typical changes: class-title changes, additions of classes, relocations of classes in the hierarchy (by moving them up or down in the hierarchy, or horizontally), relocations of a whole subtree in the hierarchy, merges of two classes (in two variants: two classes become one new class, or one class is appended to the other class), splits of a classes, and pure deletions. However, current versioning techniques for content standards are often quite simple. In UNSPSC, for example, all changes are encoded as either additions, deletions or edits (title changes). This means that the relocation of a subtree is specified as a sequence of "delete a list of classes" and "add a list of classes".

---

[5] http://eccma.org/unspsc/

[6] http://www.rosettanet.org/

[7] http://www.eclass.de/

3.2.3 Conceptual implication of changes

Another problem that is involved with ontology change is the possible discrepancy between changes in the specification and changes the conceptualisation. The actual specification of concepts and properties is a *specific* representation of the conceptualisation; however, the same concepts could also be specified differently. Hence, a change in the specification does not necessarily coincide with a change in the conceptualisation [Klein and Fensel, 2001], and changes in the specification of an ontology are not *per definition* ontological changes.

For example, there are changes in the definition of a concept which are not meant to change the concept, and, the other way around, a concept can change without a change in its logical definition. An example of the first case is attaching a slot "fuel-type" to a class "Car". Both class-definitions still refer to the same ontological concept, but in the second version it is described more extensively. On the other hand, a natural language definition of a concept might change without a logical change in the definition of a concept, e.g. a new definition of "Chair" might exclude reclining-chair.

In the literature, these different types of changes are distinguished in the following way [Visser et al, 1997b; Klein 2001]:

- a **conceptual change** is a change in the interpretation of a domain (i.e. the conceptualisation), which results in different ontological concepts or different relations between those concepts;
- a **explication change** is a change in the way the conceptualisation is specified.

It is impossible to determine the type of change automatically, because this is basically a decision of the ontology engineer. Therefore, it is necessary to allow ontology engineers to specify their intention of their change. If they characterise a change in a definition as "conceptual", then the source and target definitions should be considered as different (even when their specification is the same), else, if a change is "explicational", the two definitions can be regarded as equivalent.

3.3 Change management

A change management methodology that allows partly automatic transformation of data and ontologies between different versions is essential. Such a methodology should be able to cope with the different types of incompatibility, should allow a precise specification of changes, and should help ontology engineers to specify the conceptual consequence of the change. We will now discuss two aspects of such a methodology: a comparison tool for ontologies and a the change specification mechanism.

3.3.1 Comparing ontologies

An important aspect of a change management methodology is the ability to compare versions of ontologies and highlight the differences. This helps in finding changes in ontologies, even if those have occurred in an uncontrolled way, i.e., possibly by different people in an unknown order. In the framework of the OnToKnowledge project, an Ontology Versioning Server is being developed. The system provides a web-based system to manage changes in ontologies. Its main function is to present a transparent interface to arbitrary versions of ontologies. To achieve this, the system maintains an internal specification of the relation between the different variants of

ontologies. It allows users to differentiate between ontologies at a conceptual level and to export the differences as adaptations or transformations.

One of the central features of system is the ability to compare ontologies at a conceptual level. This is inspired by UNIX `diff`, but the implementation is quite different. Standard `diff` compares file version at line-level, highlighting the lines that textually differ in two versions. The OnToKnowledge system, in contrast, compares version of ontologies at a *structural* level, showing which definitions of ontological concepts or properties are changed.

The comparison function distinguishes between the following types of change:

- Non-logical change, e.g. in a natural language description. This are changes in the label of an concept or property, or in comment inside definitions.
- Logical definition change. This is a change in the definition of a concept that affects its formal semantics. Examples of such changes are alterations of subclass statements, or changes in the domain or range of properties. Additions or deletions of local property restrictions in a class are also logical changes. The second and third change in the Figure 3 (class “Male” and property “hasParent”) are examples of such changes.
- Identifier change. This is the case when a concept or property is given a new identifier, i.e. a renaming.
- Addition of definitions.
- Deletion of definitions.

Each type of change is highlighted in a different color, and the actually changed lines are printed in boldface. An example of the visual representation of the result of a comparison is shown in Figure 3.

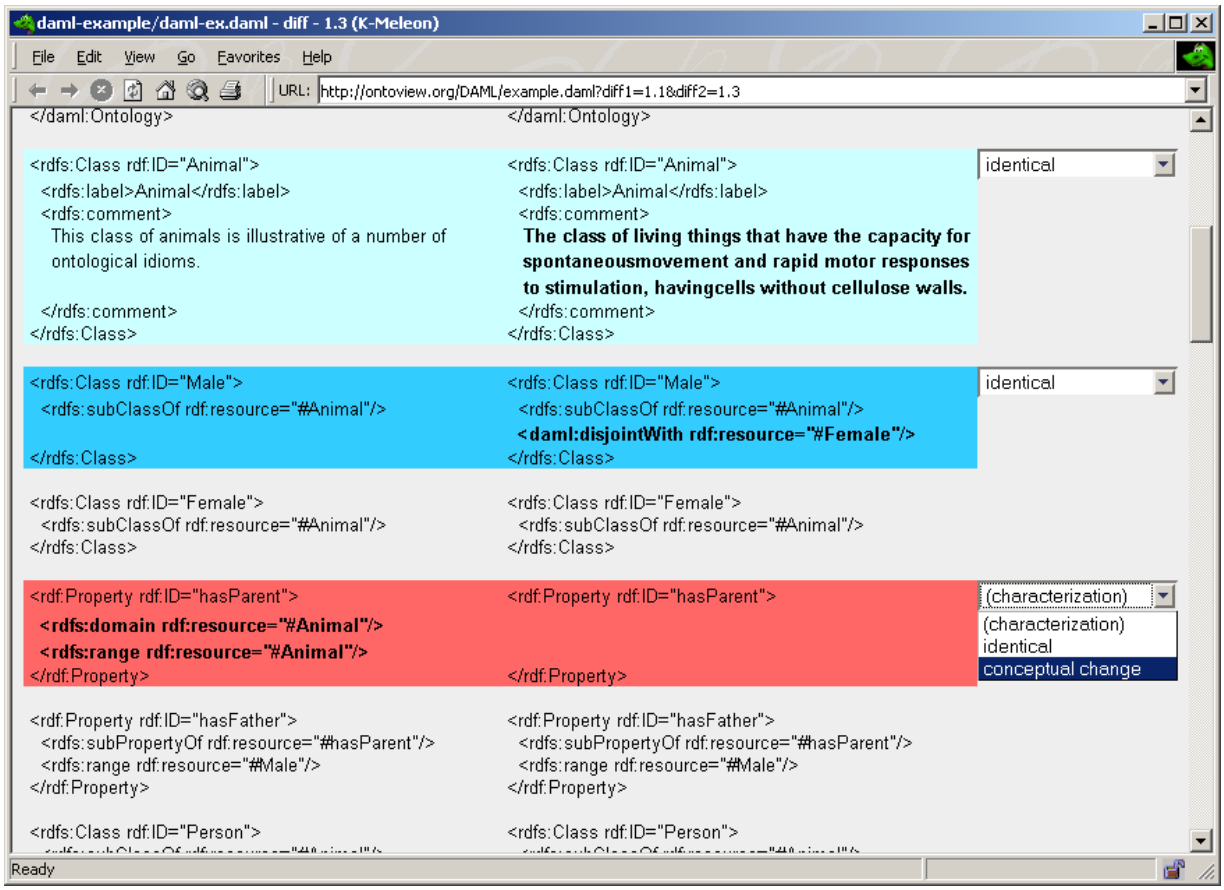

Figure 3. The result of a comparison of two ontologies.

The comparison function also allows the user to *characterize* the conceptual implication of the changes. For the first three types of changes, the user is given the option to label them either as "identical" (i.e., the change is an explication change), or as "conceptual change". In the latter case, the user can specify the conceptual relation between the two version of the concept. For example, by stating that the property "hasParent$_{1.0}$" is a sub-property of "hasParent$_{2.0}$".

Another function is the possibility to analysis effects of changes. Changes in ontologies do not only affect the data and applications that use them, but they can also have unintended, unexpected and unforeseeable consequences in the ontology itself. [McGuinness et al, 2000]. The system provides some basic support for the analysis of these effects. First, on request it can also highlight the places in the ontology where conceptually changed concepts or properties are used. For example, if a property "hasChild" is changed, it will highlight the definition of the class "Mother", which uses the property "hasChild". This function can also exploit the transitivity of properties to show the propagation of possible changes through the ontology.

3.3.2 Specification of change

A change in an ontology constitutes a new version of the ontology. This new version defines an orthogonal *update* relation between the definitions in the original version of the ontology and those in the new version. The update relation between two versions of a concept, e.g. between class $A_{1.0}$ and class $A_{2.0}$, is fundamentally different

from the relation between two concepts inside an ontology, e.g. between class A and class B. In the latter case, the relation is purely conceptually; however the update relations also has meta-information about the change of the concept associated with it.

We distinguish the following properties that are associated with an update relation:

- **transformation** or **actual change**: a specification of what has actually changed in an ontological definition, specified by a set of change operations (cf. [Banerjee et al, 1987], e.g., change of a restriction on a property, addition of a class, removal of a property, etc.;
- **conceptual relation**: the logical relation between constructs in the two versions of the ontology, e.g., specified by equivalence relations, subsumption relations, logical rules, or approximations. The conceptual relation between two versions of a concept specifies the intention of the ontology engineer that characterised the change.
- descriptive meta-data like **date**, **author**, and **reason** of the update: this describes the when, who and why of the change;
- **valid context**: a description of the context in which the update is valid. In its simplest form, this is the time-period in which the change is valid in the real world, conform to *valid date* in temporal databases [Roddick, 1995] (in this terminology, the "date" in the descriptive meta-data is called *transaction date*). More extensive descriptions of the context, in various degrees of formality, are also possible.

Keeping track of all these four aspects of a change relation serves several functions. It is possible to perform loss-less transformations of ontologies, by exploiting the set of change operations. The conceptual relation gives the ability to re-interpret data and other ontologies that use the changed ontology via the new ontology. The meta-data and context helps to select versions and validate their applicability.

4 Organising ontologies

As the number of different ontologies is increasing, the task of storing, maintaining and re-organising them to secure the successful re-use of ontologies is challenging (Fensel, 2001). *Ontology library systems* are an important tool in grouping and re-organising ontologies for further re-use, integration, maintenance, mapping and versioning. Basically it is a library system that offers various functions for managing, adapting and standardising groups of ontologies. It should be easily accessible and offer efficient support for re-using existing relevant ontologies and standardising them based on upper-level ontologies and ontology representation languages (Ding & Fensel, 2001).

4.1 Storage need in OTK

Sesame[8] – developed in the OnToKnowledge project – allows persistent storage of RDF data and RDFs information and subsequent querying of the information enabled by RQL. Sesame selected a relation database as the storage mechanism, but is DBMS-independent via the Repository Abstraction Layer (RAL). The RAL is an

[8] http://sesame.aidministrator.nl/

interface that offers RDF-specific methods to its clients and translates these methods to calls to its specific DBMS. The big advantage of RAL is that it makes it possible to implement Sesame on top of a wide variety of repositories without changing any of Sesame's other components (Broekstra et al, 2000).

Sesame provides a basis functionality to store ontologies and their instances and provide the querying service as well. However, for large-scale use and ontology re-use, more advanced functions are needed. A real-world ontology library system must support the following (see Figure 4):

- Open *storage*, *identification* and *versioning* of ontologies;
- Smooth *access* to existing ontologies and advanced support in *adapting* ontologies to certain domain and task-specific circumstances (instead of requiring such ontologies to be developed from scratch);
- Fully employing the power of *standardisation* and providing access to *upper-layer ontologies* and standard *representation languages.*

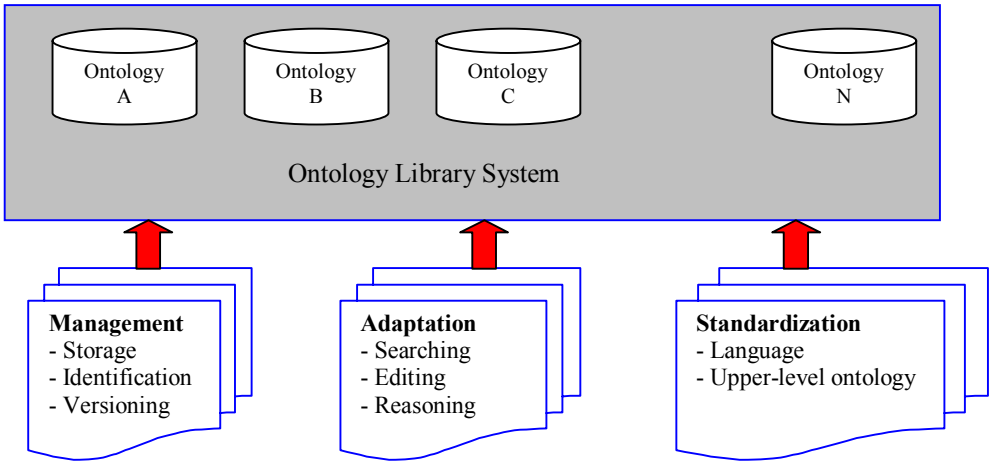

Figure 4. Aspects of an ontology library system.

4.2 Functionality of an ontology storage system

An ontology library system should feature a functional infrastructure to store and maintain ontologies, an uncomplicated adapting environment for editing, searching and reasoning ontologies, and strong standardization support by providing upper-level ontologies and standard ontology representation languages.

The aspects above can be further specified as the follows:

4.2.1 Management

The main purpose of ontologies is to enable knowledge sharing and re-use (Visser, van Kralingen & Bench-Capon, 1997). Important functions should include open storage, identification, and versioning support.

- **Storage** (how to store the ontology): (a) Is the ontology easily accessible (via a client/server architecture, Peer-to-Peer, etc.); (b) Are ontologies classified according to some existing or home-made categories; and (c) Are ontologies stored in modules? (The modularity structure can guarantees proficient ontology re-use).

- **Identification** (how to uniquely identify an ontology): Each ontology must have a unique identifier in the ontology library system.
- **Versioning** (how to maintain the changes of ontologies in an ontology library system): Versioning is very critical in ensuring the consistency among different versions of ontologies.

4.2.2 Adaptation.

Ontology library systems should facilitate the task of extending and updating ontologies. They should provide user-friendly environments for searching, editing and reasoning ontologies. Important aspects include support in finding and modifying existing ontologies.

- **Searching** (how to search ontology): Does a library system provide certain searching facilities, such as keyword-based searching or other advanced searching? Does it feature an adequate browsing function?
- **Editing** (how to add, delete and edit specific ontologies): How does the system support the editing function? Does it support remote and cooperative editing?
- **Reasoning** (how to derive consequences from an ontology): How does the system support ontology evaluation and verification? Is it possible to derive any query-answering behaviour?

4.2.3 Standardisation

Ontology library systems should support existing or available standards, such as standardised ontology representation languages and standardised taxonomies or structures of ontologies.

- **Language** (the kind of standard ontology language used in the ontology library system, for instance, RDFs[9], XMLs[10] or DAML+OIL[11]): Does the system only support one standard language or other different languages?

4.2.4 Upper-level ontologies

It can be useful if the ontology library system is 'grounded' in any existing upper-level ontologies, such as Upper Cyc Ontology, SENSUS, MikroKosmos, the PENNMAN Upper Model, and IEEE upper-layer ontology. The upper-level ontology captures and models the basic concepts and knowledge that could be re-used in creating new ontologies and in organising ontology libraries.

4.3 Current storage systems

We have surveyed a number of existing ontology library systems, to analyse the current state-of-the art in ontology library systems. The systems that we included in our survey are: WebOnto[12], Ontolingua [13], DAML Ontology library system, SHOE[14],

---

[9] http://www.w3.org/RDF/

[10] http://www.w3.org/XML/

[11] http://www.ontoknowledge.org/oil/oilhome.shtml

[12] http://eldora.open.ac.uk:3000/webonto

Ontology Server from Vrije Universiteit, Brussels, Belgium[15], IEEE Standard Upper Ontology[16], OntoServer[17] and ONIONS[18]. Not all of them real ontology library systems, but each of them provides at least some aspects of an library system. Also, there are other ontology library systems than those that we included in our comparison. We have only included approaches that are publicly available as those offer enough detailed information to enable us to evaluate their actual functionality. The surveyed results have been related to Sesame to identify the extensions that are might be required in the future. The following summarises the features of above mentioned ontology library systems.

4.3.1 Management

Storage. The ontology library systems in this survey fall into one of two categories: (a) those with a client/server-based *architecture* aimed at enabling remote accessing and collaborative editing (WebOnto, Ontolingua, DAML Ontology Library); and (b) those that feature web-accessible architecture (SHOE, IEEE SUO). Ontology Server features a database-structured architecture. Most ontologies in this survey are classified or indexed. They are stored in a modular structured library (or lattice of ontologies). WebOnto, Ontolingua and ONIONS all highlight the importance of a modular structure in an ontology library system as that structure facilitates the task of reorganising ontology library systems and re-using and managing ontologies.

Identification. The standard way to identify an ontology is by its Unique name or Identifier.

Versioning. Only SHOE supports versioning for handling the dynamic changes of ontologies. Versioning is an important aspect of the ontology library system. Although many of the systems surveyed do not currently have this function, they clearly show that it is needed for future improvements.

Sesame. It has a client/server-based architecture and supports the web access. But it doesn't support the collaborative editing. It deploys the RAL that makes it possible to implement Sesame on top of a wide variety of database-structured repositories without changing any of Sesame's other components. The ontologies stored in Sesame are not classified and also not in the modular structure either. Sesame will support ontology versioning in the near future via the collaboration with the partner in the OnToKnowledge project.

4.3.2 Adaptation

Searching. Most of these ontology library systems can be accessed through the Internet or World Wide Web. They offer simple browsing only. Ontolingua is the only one that offers some functional searching features, such as keyword searching (wide-card searching), simple query answering, context sensitive searching, etc. As it

---

[13] http://www-ksl-svc.stanford.edu:5915/

[14] http://www.cs.umd.edu/projects/plus/SHOE/

[15] http://www.starlab.vub.ac.be/research/dogma/OntologyServer.htm

[16] http://suo.ieee.org/refs.html

[17] http://ontoserver.aifb.uni-karlsruhe.de/

[18] http://saussure.irmkant.rm.cnr.it/onto/

is embedded in the database management system, Ontology Server could also provide SQL-based searching.

Editing. Most ontology library systems only provide simple editing functions. WebOnto and Ontolingua support collaborative ontology editing (asynchronous and synchronous).

Reasoning. Very simple reasoning functions are provided by WebOnto (rule-based reasoning), Ontolingua (ontology testing) and SHOE (ontology revision).

Sesame. It supports the highly expressive querying of RDFs and RDF data enabled by RQL. It has some simple editing functions but not the collaborative editing. The reasoning function of Sesame will be developed via the OnToKnowledge Project.

4.3.3 Standardisation

Language. These ontology library systems use different languages to store their ontologies. In this case, the important function for the future ontology library system should support inter-language translating (like Ontolingua) or some standard language should be accepted or proposed within the ontology community (such as DAML+OIL).

Upper-level Ontology. Ontolingua has a public version of CYC upper-level ontology called HPKB-UPPER-LEVEL with some modification drawn from Pangloss, WordNet, and Penma. WebOnto and SHOE doesn't have the standard upper-level ontology but has its own fine-grained structure (e.g., Base Ontology). IEEE SUO tries to set up a public standard upper-level ontology.

Sesame. It supports RDF, RDFs and DAML+OIL. It doesn't have any upper-level ontology.

4.4 Requirements for storage system in OTK

In this part, we will summarise important requirements for structuring an ontology library system to enhance ontology management, adaptation and standardisation. Thus doing, we will formulate a wish-list for an ideal ontology library system.

4.4.1 Management

Storage. A client/server-based *architecture* is critical to an ontology library system's capacity to support collaborative ontology editing. An ontology library system should also be web accessible.

It is necessary to *classify* ontology in an ontology library system in order to facilitate searching, managing and re-using ontology. Some of the ontology classification mechanisms available are based on distinguishable features of ontologies. Examples include the following:

- the *subject* of ontologies (The DAML ontology library system classifies ontologies according to the Open Directory Category (www.dmoz.org));
- the *structure* of the ontology (The Ontolingua ontology library system has an inclusion lattice showing the inclusion relations between different ontologies);
- inter and intra ontology *features* (Visser and Bench-Capon (1998) indexed ontologies based on the intra and inter ontology features. Examples include

general, design process, taxonomy, axioms, inference mechanism, application, contributions, etc.);

- the *lattice* structure (Noy & Hafner (1997) built a lattice of ontologies showing the relevance of ontologies);
- the *dimensions* of the ontology (Heijst, Schreiber, and Wielinga, (1997) indexed ontologies using dimensions (task/method dependency and domain dependency) to partition the library into a core library and a peripheral library);
- *stratified upper-level* ontology (ONIONS used generic, intermediate and domain layer to index ontologies),
- the *relations* of ontology (Visser and Bench-Capon (1998) indexed ontology based on defined relations, such as the subset/superset relation, extension relation, restriction, and mapping relation),
- the *components* of ontology (Visser and Bench-Capon (1998) also mentioned the indexing of ontology based on the component of ontologies, such as domain partitioning (partition domain in logical units), alternative domain views (polymorphic refinement), abstraction (abstract and detailed ontologies), primary ontologies versus secondary ontologies, terminological, information and knowledge modelling ontologies).

Modular organisation in the ontology library system organises units into modules. This serves to maximise cohesion within modules and minimise interaction between modules (McGuinness, Fikes, Rice, & Wilder, 2000). Most of the ontology library systems that aim to facilitate ontology re-use, ontology mapping and integration have adopted this structure. ONIONS also highlights the *stratified* design of an ontology library system. Different *naming policies* assist the ontology library system to achieve the modular organisation or stratified storage of ontologies (McGuinness, Fikes, Rice, & Wilder, 2000). The disjointed partitioning of classes can facilitate modularity, assembling, integrating and consisting checking of ontologies. If, for instance, a certain class, such as 'people,' were disjointed from another class, say 'countries', then consistency checks could be carried out much sooner and faster. Thus, the partition modification has proven to be extremely valuable for editing purposes. Linking class names with their own contexts or using name space for differentiating them can serve to prevent violation within individual ontologies. As ontologies continue to grow, so too does the importance of systematic and consistent naming and organisational rules.

Identification. Unique ontology URL, Identifier and name are used as the identifier for ontologies in the ontology library systems.

Versioning. A version control mechanism is very important to an ontology library system. Unfortunately, most existing ontology library systems cannot support it, except for SHOE.

4.4.2 Adaptation

Searching & Editing. An ontology library system should feature a visualised browsing environment, using hyperlinks or cross-references to closely related information. It should support collaborative editing and offer advanced searching

features by adopting various existing information retrieval techniques, database searching features, or AI heuristic techniques. Ontology library systems could also monitor user profiles based on access patterns in order to personalise the view of ontologies (Domingue & Motta, 1999).

Reasoning. A simple reasoning function should be included in order to facilitate ontology creation, ontology mapping and integration.

4.4.3 Standardisation

Language. Syntactically, an ontology representation language should be standardised or inter- or intra- ontology language translation should be supported. Semantically, an ontology library system should feature the *common vocabulary* (or faceted taxonomy). At any rate, it should eliminate the implicitness and misunderstanding of terms in different ontologies (due to synonyms, homonyms, etc.) for most generic classes. Preferably, an ontology library system should also support compatibility with or mapping between multiple controlled vocabularies from different domains. This would not only serve to guarantee flexibility in expressing an ontology semantically, but also to liquidate implicitness. The structures of these common vocabularies or multiple controlled vocabularies must be faceted, or modulated so as to facilitate the re-use, mapping and integration of ontologies (McGuinness, 2000). These vocabularies can help in simple synonym matching, sibling analysis, and disjoint partition checking.

Upper-level Ontology. Standard upper-level ontology is important for better organisation of ontology library systems (Ontolingua, IEEE SUO).

4.4.4 Others

Ontology scalability. Ontology library systems should also consider increasing the scale of ontologies.

Maintaining facility. Ontology library systems should also provide some maintenance features, such as consistency checking, diagnostic testing, support for changes, and adaptation of ontologies for different applications.

Explicit documentation. Each ontology in an ontology library system should be extensively documented. The documentation should include such information as how the ontology was constructed, how to make extensions and what the ontology's naming policy, organisational principles and functions are. Such explicit documents about the ontologies themselves will pave the way for efficient ontology management and re-use.

5 Summary

In this chapter, we looked at various aspects of ontology management. Ontology management is the whole set of methods, methodologies, and techniques that is necessary to efficiently use multiple variants of ontologies from possibly different sources for different tasks.

Alignment is an important aspect, because in many real-world scenario's, there are several ontologies of a domain used for a specific task. Each of those domain ontologies might capture specific aspects of knowledge and might use different terminology. Special mapping ontologies must be created to link different

terminologies and modelling styles used in these domain specific ontologies. We described a meta-ontology that can be used to creating such bridges between separated pieces of knowledge. These bridges along with domain ontologies can then used to perform cross-ontology tasks.

We also discussed the fact ontologies are not static, but evolve over time. Domain changes, adaptations to different tasks, or changes in the conceptualisation require modifications of the ontology. The evolution of ontologies causes interoperability problems which might hamper their effective reuse. Ontology comparison techniques can help the ontology engineer to find changes between ontologies and to characterise them conceptually. When the conceptual relation between the versions, the transformations between them, as well as the meta-data of the change is maintained, it is possible to support both loss-less transformations between version *and* re-interpretation of data and knowledge under different versions.

Ontology library systems are systems that support the ontology management task in various aspects. We have discussed the functions of a ontology library system, we surveyed exiting systems and finally came up with a wish-list for the ideal system.

References

Banerjee, J., Kim, W., Kim, H.-J., and Korth, H. F. (1987). Semantics and Implementation of Schema Evolution in Object-Oriented Databases. *SIGMOD Record (Proc. Conf. on Management of Data)*, 16(3):311–322, May 1987.

Broekstra, J., Fluit, C., and van Harmelen, F. (2000), "The State of the Art on Representation and Query Languages for Semistructured Data", *IST-199-10132 On-To-Knowledge Project*, Deliverable 8, 2000. http://www.ontoknowledge.org/del.shtml

Clark, J. (1999). XSL Transformations (XSL-T), W3C Recommendation, 1999. [http://www.w3.org/TR/xslt/]

Corcho, O. and Gomez-Perez, A. (2000) 'A roadmap to ontology specification languages', in Knowledge Engineering and Knowledge Management; Methods, Models and Tools, Proceedings of the 12th International Conference EKAW 2000, eds., Rose Dieng and Olivier Corby, LNCS 1937, pp. 80–96, Juan-les-Pins, France, October 2–6, 2000.

Ding, Y. and Fensel, D. (2001). Ontology Library Systems: The key for successful Ontology Reuse. The first Semantic web working symposium (SWWS1), Stanford, USA, July 29th-August 1st, 2001.

Domingue, J. & Motta, E. (1999). A knowledge-based news server supporting ontology-driven story enrichment and knowledge retrieval. In D. Fensel and R. Studer (editors), *Proceedings of the 11th European Workshop on Knowledge Acquisition, Modeling, and Management (EKAW '99),* LNAI 1621, Springer-Verlag, 1999.

Fensel, D. (2001). *Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce*. Springer, 2001.

Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2), 1993.

Heijst, G. van Schreiber, A.T. and Wielinga, B. J. (1997). Using explicit ontologies in KBS development. *Int. Journal of Human-Computer Studies* **45** 183-292.

Klein, M. (2001). 'Combining and relating ontologies: an analysis of problems and solutions', in *Workshop on Ontologies and Information Sharing, IJCAI'01*, eds., Asuncion Gomez-Perez, Michael Gruninger, Heiner Stuckenschmidt, and Michael Uschold, Seattle, USA, August 4–5, 2001.

Klein, M. and Fensel, D. (2001). Ontology versioning for the Semantic Web. In *Proceedings of the International Semantic Web Working Symposium (SWWS)*, Stanford University, California, USA, July 30 – Aug. 1, 2001.

Lassila, O. and Swick, R., (1999). Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation, 1999. [http://www.w3.org/TR/REC-rdf-syntax/]

Lerner, B.S. (2000). A Model for Compound Type Changes Encountered in Schema Evolution. *ACM Transactions on Database Systems*. **25**(1): p. 83-127, 2000.

McGuinness, D.L. (2000). Conceptual modelling for distributed ontology environment. In *Proceedings of the Eighth International Conference on Conceptual Structures Logical, Linguistic, and Computational Issues (ICCS2000), Darmstadt, Germany, August 14-18, 2000.*

McGuinness,. D.L Fikes, R. Rice, J. & Wilder, S.(2000). An environment for merging and testing large ontologies. *Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR2000)*. Breckenridge, Colorado, April 12-15, 2000.

Mitra, P., Wiederhold, G., and Kersten, M. (2000). A Graph-Oriented Model for Articulation of Ontology Interdependencies, In: Zaniolo, C., Lockemann, P., Scholl, M., and Grust, T. (eds.), Advances in Database Technology - EDBT 2000, 7th International Conference on Extending Database Technology, Springer-Verlag, LNCS 1777, Konstanz, Germany, March 27-31, 2000, pp. 86-100.

Noy, F. N. & Hafner, C.D. (1997). The state of the art in ontology design: A survey and comparative review. *AI Magazine* **4** 53-74.

Omelayenko, B. and Fensel, D. (2001). A Two-Layered Integration Approach for Product Information in B2B E-commerce, In: Madria, K. and Pernul, G. (eds.), Proceedings of the Second International Conference on Electronic Commerce and Web Technologies (EC WEB-2001), Springer-Verlag, LNCS 2115, Munich, Germany, September 4-6, 2001, pp. 226-239. [http://www.cs.vu.nl/~borys/papers/EC-Web01.pdf]

Roddick, J.F. (1995). A survey of schema versioning issues for database systems. *Information and Software Technology*. **37**(7): p. 383-393, 1995.

Uschold, M., Healy, M., Williamson, K., Clark, P. and Woods, S. (1998) 'Ontology reuse and application', in *Formal Ontology in Information Systems (FOIS'98)*, ed., N. Guarino, Treno, Italy, (June 6-8, 1998). IOS Press, Amsterdam.

Ventrone, V. and Heiler, S. (1991). Semantic heterogeneity as a result of domain evolution. *SIGMOD Record (ACM Special Interest Group on Management of Data)*. **20**(4): p. 16-20, 1991.

Visser, P.R.S. and Bench-Capon, T.J.M (1998). A Comparison of Four Ontologies for the Design of Legal Knowledge Systems. *Artificial Intelligence and Law* **6** 27-57.

Visser, P.R.S., van Kralingen, R.W. & Bench-Capon, T.J.M. (1997a). A method for the development of legal knowledge systems. *Proceedings of the Sixth International Conference on Artificial Intelligence and Law (ICAIL'97)*, Melbourne, Australia. 1997.

Visser, P. R. S., Jones, D. M., Bench-Capon, T. J. M., and Shave, M. J. R. (1997b). An analysis of ontological mismatches: Heterogeneity versus interoperability. In *AAAI 1997 Spring Symposium on Ontological Engineering*, Stanford, USA, 1997.