Semantic Information Integration Inside and Across Organizational Boundaries

Master thesis

Jos de Bruijn

Multimedia Databases, Data and Knowledge Systems group,
Faculty of Electrical Engineering, Mathematics and
Computer Science, Delft University of Technology,
Mekelweg 4, 2628 CD Delft, The Netherlands

Digital Enterprise Research Institute (DERI),
www: http://deri.semanticweb.org/

email: jos.de-bruijn@deri.ie

October 2003

Preface

This Master thesis is the final subject of my study of Technical Informatics at the faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology (TU Delft). This work was supervised from the side of the TU Delft by Dr. ir. Johan Ter Bekke of the Multimedia Databases group, part of Data and Knowledge Systems group, headed by Prof. dr. Henk Koppelaar.

I have written this Master thesis during my work at the Digital Enterprise Research Institute (DERI), location Innsbruck - Institute for Computer Science, University of Innsbruck, where I worked in the European Commission funded research projects COG (Corporate Ontology Grid) and Esperonto. This Master thesis is partly based on my work in these two projects. DERI is headed by dr. Dieter Fensel, who also supervised this Master thesis.

First a note about DERI. When I initially started to work on my Master thesis, DERI did not exist yet and I worked at the Next Web Generation research group (headed by Prof. Dieter Fensel) at the University of Innsbruck. Not long before I finished this thesis, the DERI institute was created with a location in Galway, Ireland and a location (consisting of the Next Web Generation research group) in Innsbruck, Austria.

What brought me to DERI is its state of the art research in Semantic Web technologies. Finally, the subject was chosen because of the relationship with database systems; the subject of this thesis was initially limited to database integration using Semantic Web technologies, but was later expanded to include integration of other information sources across the Web.

I would like to thank Dr. Johan ter Bekke and Prof. Dieter Fensel for supervising this Master thesis and for providing me with useful feedback. I would like to thank Prof. Dieter Fensel and all members of the Digital Enterprise Research Institute (and especially the members in Innsbruck) for providing me with such a stimulating research environment for conducting the work on my thesis. Finally, I would like to thank Prof. Dieter Fensel for giving me the opportunity to continue my research at DERI after my Master thesis and work towards a PhD.

Jos de Bruijn Innsbruck, 31 October 2003

Abstract

Information integration within enterprises is hindered by differences in software and hardware platforms and by syntactic and semantic differences in the schemas of the various heterogeneous data sources. This is a well-known problem in the area of Enterprise Application Integration (EAI), where many applications have been developed for the purpose of information integration. Most current tools, however, only address the problems of (soft- and hardware) platform and syntactic heterogeneity. The problem of semantic heterogeneity (i.e. the differences in meaning of concepts) is, in current solutions, reduced to syntactic rewriting without explicating the meaning of the data.

When broadening the scope of the information integration problem to cross organizational border, the problem becomes even more severe, because within an organization there is a certain amount of control over the applications, while there typically is no control over the applications used by other organizations or even organizational units. Standards are arising to mitigate these problems of inter-operability between organizations, but current approaches are very specific for certain branches and are typically very rigid in what they require from anyone using the standard.

We investigate these problems of semantic information integration and their solutions using ontology technology in the context of two major European research projects.

In the COG (Corporate Ontology Grid) project we try to provide a solution for the information integration problem within enterprises. We used the Unicorn Workbench tool to create an Information Model (an ontology) with a well-defined meaning and used the Workbench to map data schemas taken from the automotive industry to the Information Model in order to make the meaning of the concepts in the data schemas explicit.

In the context of the Esperonto project, we take a broader approach and try to solve the problem of Information Integration between different enterprises. We do this in the context of the Semantic Web, where we assume the enterprises are entities on the Semantic Web that have formal explicit models for their data in the form of ontologies. We introduce a solution to the information integration problem, through the creation of explicit mappings between ontologies, in a semi-automatic manner.

Semantic Web technology, and especially ontology technology, enables information integration on a semantic level both within and across organizational boundaries, but there are still many issues to resolve, such as standardization in languages for the specification of mapping rules and the development of usable, industrial-strength tools for the creation and operationalization of these mappings.

Organization

Supervising Committee

Dr. ir. Johan ter Bekke (Delft University of Technology)
Prof. dr. Dieter Fensel (Digital Enterprise Research Institute)
Prof. dr. Henk Koppelaar (Delft University of Technology)
Dr. ir. Kees van der Meer (Delft University of Technology)

Glossary

A

Asset See *External Asset*.

Atom A generic term referring to a Column, Property, Attribute, etc. (the atomic data element) in an external asset in the Unicorn Workbench.

B

Business Rule A rule applied to the values of properties of instances of classes in the Information Model of the Unicorn Workbench, relating the values of different properties or relating the value of a property to predefined values.

C

Class A class describes a set of entities with common properties, analogous to the class concept in object-oriented programming. Each entity in this set is called an *instance* of the class.

COG Project The Corporate Ontology Grid Project. A European Commission funded project that researches the use of ontology technology for Enterprise Application Integration, and, more specifically, data source integration.

Composite A generic term referring to a Table, ComplexType, Entity, etc. in an external asset. A composite aggregates a set of **Atoms**.

D

Data model See 'Data schema'.

Data schema A description of the structure of a set of data, usually residing in a single data source.

Data source A repository containing data. This data is typically domain- or application-specific. The structure of the data is described by a data schema.

Database A database is a collection of facts, usually structured according to some schema. A database is usually described by a **data schema** and stored in a Data Base Management System (**DBMS**). The model most commonly used for database schemas is the **Relational Model**; other approaches include the **Semantic data model**.

DBMS A Data Base Management System (DBMS) stores data in data bases according to the data schema specified for each specific data base and performs query processing.

E

Esperonto A European Commission-funded project that aims to bridge the gap between the current Web and the Semantic Web. The interest in the project in this thesis is the alignment of (and mapping between) models (ontologies) used to annotate data on the Semantic Web.

External Asset A data source, whose data schema is imported into the Unicorn Workbench.

External Asset Layer The layer in the Unicorn Workbench that contains the imported data schemas and the mappings between these schemas and the Information Model.

F

Facet Facets describe properties of **slots**, such as cardinality (i.e. stating how many values can be assigned to the property, zero, one, or more).

Frame A frame is a data structure that is typically used to represent a single entity and the slots and facets associated with it. A frame can be either a class or an individual (an instance of a class).

G

Global data schema A data schema describing the data from a collection of data sources. Such a global schema can be an aggregation of local schemas or it can be built from scratch.

I

Information Model The central ontology in the Unicorn Workbench, containing all the packages, classes, properties, and business rules.

Instance A thing (or being) in the real world. An instance is a member of a class the properties of the instance are sufficiently described by the class.

Inverse properties Two properties are inverses of each other if the source of the first property is equivalent to the type of the second property and vice versa. For example, the property `hasWrittenBook` of class `Author` would be the inverse of the property `writtenBy` of class `Book`. The inverse relationship is symmetrical.

L

Local data schema A data schema for a specific data souce or a specific user or agent.

M

Medadata Metadata is data about data. Metadata can say something about the meaning and/or the structure of data. Another possible interpretation of metadata is information that says something about the history, the authoring or the classification of the data. Examples here are the author, revision date, keywords, etc. . . .

Model Layer The Model Layer in the Unicorn Workbench consists of the Information Model. It functions as a mediator between the user applications and the data sources, aggregated in the **External Asset Layer**.

O

Ontology An explicit formal specification of a consensual domain theory; an *Information Model* created in the Unicorn Workbench is an ontology. Ontologies interweave human and machine understanding in order to enable *knowledge sharing and reuse* between humans and computers.

Ontology Alignment Ontology Alignment is the process of relating (syntactically and/or semantically) different ontologies, which have some semantic overlap (i.e. describe, partly or wholly, the same domain), to each other through an ontology mapping.

Ontology Mapping Ontology mapping is the declarative specification of a set of mapping rules, which relate two distinct ontologies to each other.

Ontology Merging Ontology Merging is the process of creating one target ontology from several distinct, though semantically overlapping, source ontologies.

P

Property A property of a class represents an attribute shared by a group of instances.

R

RDBMS A Relational Data Base Management System (RDBMS) is a Data Base Management System (**DBMS**) that uses the **Relational Model** as its data model.

Relational Model A data model for database schemas. It represent the database as a collection of *relations*.

S

Semantic data model A data model for database schemas. Provides a much richer formal and real-world semantics than the **Relational Model**.

Slot A slot defines a relation between two classes.

Source of a property The class to which the property belongs.

Source schema The **data schema** for an **external asset** used as a source for Information Integration.

Subclass A class that describes a collection of instances that is a subset of the collection of instances of the superclass. This also means that all the properties and business rules of the superclass are inherited by the subclass.

Superclass A class that describes a collection of instances that is a superset of the collections of instances of all its subclasses.

T

Test Instance A fictive **instance** created in the **Unicorn Workbench** in order to help validate the **Information Model**.

Type of a property Specifies the range of values that can be assigned to the property.

U

Unicorn Workbench The semantic data integration tool created by Unicorn. Also referred to simply as 'Unicorn' and 'Workbench'.

Contents

Chapter 1

Introduction

Continually, new applications are introduced in enterprises, while existing legacy applications cannot be replaced because of the investments that have been made in the past. These new applications need to use data residing in the legacy applications. This data is typically stored in a proprietary format. If a new application is to reuse the information residing in legacy systems, this information needs to be made available in a way that is understandable by the new application. Not only differences in hard- and software platforms need to be overcome, but also differences in syntax and semantics.

We are concerned here with the differences in semantics of these existing data schemas. The syntax is the way the data model has been written down, whereas the semantics is the intended meaning of the concepts in the data schema. It is especially hard to overcome these differences in semantics, because these semantics are often informal; the meaning depends on the name chosen for the concept. Now, such a name may be interpreted differently by different people, so that the meaning is ambiguous. This is where ontologies come into play.

The term ontology originally comes from philosophy, but has been adopted by several AI (Artificial Intelligence) research communities, originally the knowledge engineering, natural-language processing, and knowledge representation communities. In the late 1990s the notion of ontologies also became widespread in fields such as intelligent information integration, information retrieval on the Internet, and knowledge management [Studer et al., 1998].

In AI an ontology refers to a description of a part of the world in a program. Ontologies were developed to facilitate knowledge sharing and reuse [Fensel, 2003]. An important definition of ontology, used by many researchers in the field of ontologies is given by Gruber in [Gruber, 1993]: "An ontology is a formal explicit specification of a shared conceptualization." A conceptualization is an abstract simplified view of the world that we wish to represent for some purpose. The ontology is a specification because it represents the conceptualization in a concrete form. It is explicit because all concepts and constraints used are explicitly defined. Formal means the ontology should be machine understandable. Shared indicates that the ontology captures consensual knowledge [Studer et al., 1998].

A reason for the increasing interest in ontologies nowadays comes from the development of the Semantic Web [Berners-Lee et al., 2001]. Tim Berners-Lee,

inventor of the current World Wide Web and director of the World Wide Web Consortium (W3C), envisions the Semantic Web as the next generation of the current Web, where the information will be machine-readable and the services will be automated. According to [Fensel, 2003] "The explicit representation of the semantics underlying data, programs, pages, and other web resources will enable a knowledge-based web that provides a qualitatively new level of service." Ontologies provide such an explicit representation of semantics.

In the following section, we will introduce the problem of Semantic Information Integration within the enterprise as we face it today. Then we will briefly give an overview of the generic approaches in Information Integration (and especially Semantic Information Integration).

1.1 The Problem of Semantic Information Integration

What do we understand by Semantic Information Integration? We distinguish several levels on which we can do Information Integration, that is the integration of data models in order to enable inter-operation between applications[1]; we distinguish:

- The *hardware* (platform) level. This encompasses differences in computer hardware, networks, etc...
- The *software* (platform) level. This encompasses differences in Operating System (OS), database platform, etc...
- The *syntactical* level. This encompasses the way the data model is written down. For example, in one data schema there may be a concept 'EMP#' and in another schema the concept 'EmpNr'. If exactly the same thing is meant by these two concepts (e.g. employee number), then we say that there is only a syntactical difference; it can be resolved by syntactical rewriting of the schema.
- The *semantic* level. This level encompasses the intended meaning of the concepts in a schema. It is often hard to know what the intended meaning of a concept is in a "weak" schema language, such as relational algebra. Here, the meaning of the concepts depends mostly on its name, which can be interpreted differently by different people. For example, an engineer might consider a peace of metal at the start of a automobile assembly line already a car, while the average person would only see an undefined peace of metal.

By taking on the problem of Semantic Information Integration, we take on a problem that is not addressed by current solutions for Enterprise Application Integration (EAI). These solutions still require a human user to define a mapping between the message types of any two applications, where the user just

[1] Note that we focus here on only one aspect of application integration, namely the integration of data. Other aspects such as the actual exchange of messages between applications and such things as business process integration are left out, since they are not within the scope of this thesis.

provides rules to rewrite the message. In other words, current EAI solutions just employ syntactic rewriting on the basis of some predefined rules to enable inter-operation between applications. The challenge is to find a way to express the data schemas in such a way that the definition of a term is unambiguous and expressed in a formal and explicit way, linking human understanding with machine understanding.

Currently, when an application needs to use data from another application, a specific transformation script is written to transform the data from one format to the other. Such a transformation is hard to maintain, because when one of the formats changes, that change needs to be detected and the transformation needs to be updated accordingly. This problem gets even harder when many such transformations exist in an enterprise. We refer to this kind of integration as ad-hoc information integration.

This type of ad-hoc integration is not scalable, because for each two data sources that have to be integrated a custom transformation script or program needs to be built and maintained. The latter is the hardest problem, because if a data schema is changed, all transformation scripts with this schema as its source or its target need to be updated, which is a very big maintenance task; furthermore, it is easy to overlook one or two transformations if there are so many of them.

When integrating information on the semantic level, an explicit mapping is created between concepts in the different data models. Furthermore, when creating a mapping between a concept expressed in a more expressive formal language and a concept expressed in a less expressive, less formal language, the latter concept inherits the explicit semantics of the former concept. Thus, by creating such a mapping, concepts in for example a database schema can be given a formal meaning by linking them to concepts in an ontology (where, if the ontology is modelled correctly, the meaning of the concepts is specified in a formal and explicit way). We see an application of this type of mappings (between data schemas and ontologies) in the COG project (see chapter 2).

While the mapping between data schemas and ontologies is a solution for the Semantic Information Integration problem in an enterprise, it does not solve the problems of integration between organizations. An organization would not want to share its internal knowledge with other organizations; knowledge is a very valuable thing and enterprises do not want to share it. Another problem with information integration based on data schemas between organizations is the fact that the inter-organization environment is very open; when directly integrating database schemas, maintenance would become nearly impossible. We envision the use of ontologies to enable knowledge sharing and exchange between organizations over the Internet (or rather the Semantic Web). A company can determine the extent of the knowledge it wants to share, for example to perform certain business transactions, such as purchasing or selling goods. We can imagine that after data schemas in an organization are integrated using ontology technology, as is done in the COG project, the enterprise will determine a subset of the ontology that it wants to share with the world or with certain business partner. When this ontology is now shared with others on the Semantic Web, mappings need to be made between different ontologies from different organizations in order to allow inter-operation. In the Esperonto project (chapter 5) we address this problem of mapping between ontologies. In this project we address the technology necessary to create these mappings.

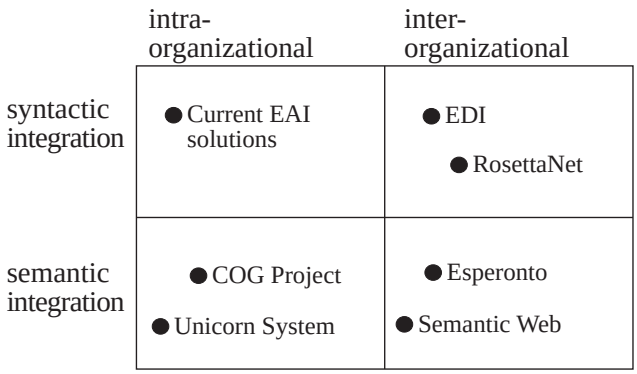

Figure 1.1: Categorization of information integration solutions

Figure 1.1 illustrates the different types of information integration solutions. We distinguish between syntactic and semantic integration and between intra- and inter-organizational integration of information. The current syntactic integration that is being done, consists of the current Enterprise Application Integration (EAI) tools, which are nowadays used to integrate applications within organizations. Inter-organizational integration is currently being done on the basis of agreements of message formats between individual organizations. The old and costly EDI (Electronic Data Interchange) standard is still heavily in use, as well as the upcoming XML standard. Both standards only dictate the way messages are exchanged; not the content of the messages. Several content standards have arisen, such as RosettaNet (http://www.rosettanet.org). These standards prescribe the format of messages to be exchanged between organizations in certain market segments. All companies adhering to this standard can now participate in the message exchange.

The approaches to information integration presented in this thesis fall in the categories of semantic intra-organizational and inter-organization integration, respectively. The COG project, presented in chapter 2, addresses information integration inside the organization, by mapping application data sources in the organization to a semantically rich central data model (the ontology). In the context of the Esperonto project, we address the integration of information between organizations using ontologies as a basis.

We can distinguish two other dimensions of information integration. We address here both the design-time and the run-time dimension of the Semantic Information Integration problem. During the design-time the mapping between the different data models is created, while during the run-time, actual data are translated from one data source to another.

1.2 Comparing ontologies and (conceptual) database models

In this section we compare ontologies with (conceptual) database models.

We will first introduce databases and ontologies, after which we introduce the specific database modelling languages to be compared, namely the Entity-

STUDNAME	ADDRESS	EMAIL	TYPE
John Smith	23, Foostreet, Bartown	jsmith@foo.bar	PhD
Mary Jansen	4, Foostreet, Bartown	mjansen@foo.bar	Msc
Pete Blah	1, Blahstreet, Bartown	pblah@foo.bar	Msc

Table 1.1: Example table 'Student'

Relationship model, the Relational Model and the Semantic Database model and the ontology specification language OWL (DL).

1.2.1 Introducing database models

A database is a collection of facts (data), usually structured according to a certain schema.

Most modern database systems structure the data in tables, where each row (also called 'tuple') represents a single entity and the values in the columns are the attributes of the entities. An example database table is given in Table 1.1. In this table, attributes of three students, namely John Smith, Mary Jansen and Pete Blah are enumerated. The attributes are the student's name, the student's address, the email address and the type of student (Msc or PhD). Each row in the table must be unique and each row is identified (in the relational model) by all its attribute values. Because this is not desirable, usually some restrictions are put on the identification of entities and a *primary key* is created. A possible primary key for the table in the example is the column 'STUDNAME'. If such a primary key is introduced, the attribute values in the primary key columns now uniquely identify the entity. So in this case, there could be no two students with the name 'John Smith'. Obviously, it can happen in a real-life database that two students have the same name, so one must be careful in choosing the primary key. In the subsection on the relational model, we discuss this and other shortcomings of this way of modelling databases.

In the database community, usually three different types of schemas are distinguished in the three-schema architecture for database systems. The *three-schema architecture* (illustrated in Figure 1.2), also known as the ANSI/SPARC architecture [Tsichritzis and Klug, 1978], introduces these three layers of database schemas [Elmasri and Navathe, 2000]:

External schemas The external schemas are application-specific views of the data model. An external schema is created for a certain user group or a certain application and captures part of the database, namely that part that is of interest to a certain user group.

Conceptual schema The conceptual schema describes the structure of the whole database for a community of users. The conceptual schema hides the physical storage structures and thus provides a model that is more understandable to the human user.

Internal schema The internal schema is a translation of the conceptual schema into a format that conforms with the actual database platform; the internal schema is directly used for the storage of data and therefore often contain some optimization that make them less readable for a human.

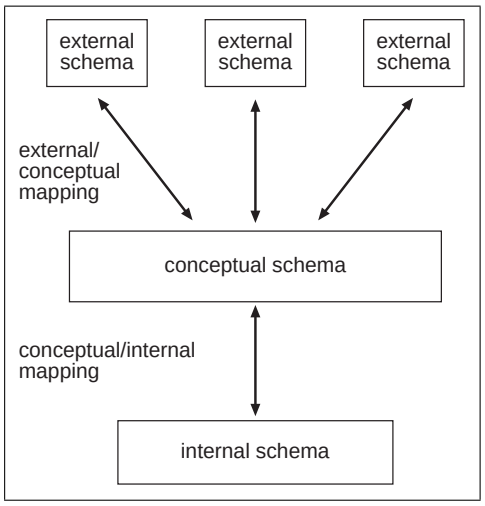

Figure 1.2: The Three-schema architecture for databases

The three-schema architecture has been introduced as an aid to achieve *data independence*, which is the capacity to change the database schema at one level, without affecting the schema at the next higher level [Elmasri and Navathe, 2000]. We distinguish two kind of data independence:

Logical data independence When logical data independence is achieved, the conceptual schema can be changed without having to change the external schemas. This means that whenever concepts are added to the conceptual schema, external schemas should not have to be changed. Whenever concepts are deleted from the conceptual schema and these concepts are not mapped to a particular external schema, the external schema does not have to be changed.

Physical data independence When physical data independence is achieved, the internal schema can be changed without affecting the conceptual schema. For example, if the file structure inside the database is changed, it should not effect the conceptual view of the data.

The three-schema architecture is referenced in many places, but it is not implemented in many commercial database systems today, because the two levels of mappings create a big overhead and degrade performance in query processing.

We will concern ourselves in the remainder mainly with the conceptual schema level, because in this area most overlap between database schemas and ontologies occurs. In fact, a conceptual database schema can be seen as an ontology and an ontology can be seen as a conceptual database schema.

An example of a data model often used for high-level conceptual schemas is the Entity-Relationship (ER) model [Chen, 1979] and the Enhanced Entity-Relationship (EER) model. The ER model is commonly in use, because of the correspondence with the relational model, which is the most used data model for database systems, and the known translations from ER Diagrams to Relational models (cf. [Elmasri and Navathe, 2000]).

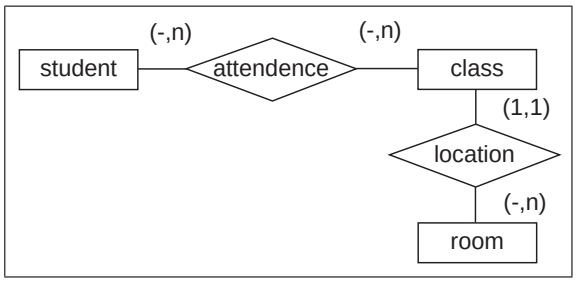

Figure 1.3: Example Entity-Relationship Diagram

Database schemas on the conceptual level can be compared with ontologies. They concern the same level of abstraction; they both aim to model a domain in a non-application and non-implementation dependent way, even though conceptual database schema are often necessarily application-dependent when one views the database, or rather the information system for which the database is created, as an application. However, many current approaches in conceptual data modelling, such as ER (the Entity Relationship model), lack the expressiveness and formal, real-world semantics of ontology languages. We see later on in this chapter that there are other approaches in conceptual database modelling with more expressivity and formal and real-world semantics.

An example Entity-Relationship model is given in Figure 1.3. For reasons of simplicity, we left out the attributes of the entity types and focussed on the entity types and the relationships. Entity types are depicted by rectangles, whereas relationships are depicted with diamond shapes.

The model consists of the following entity types: student, class, and room. There is a relationship 'attendance' between the entity types 'student' and 'class'. An arbitrary number[2] of students and an arbitrary number of classes can participate in this relationship. This means, a student can attend and arbitrary number of classes and a class can be attended by an arbitrary number of students. The relationship 'location' between 'class' and 'room' depicts the location where a class is held. A class is held in exactly one room (hence the (1,1) which means no less and no more than one) and in each room an arbitrary number of classes is held (i.e. there can be rooms where no classes are held and rooms where several classes are held).

Even though the ER data model seems relatively simple (in the simple case just consisting of entities and relations), ER models can become quite complex, even for relatively simple domains (cf. [ter Bekke, 1997b]).

1.2.2 Relational database modelling

Relational databases are semantically grounded in relational algebra [Ullman, 1988]. Furthermore, relational databases use SQL (Structured Query Language) for both the data definition and the data manipulation[3].

[2] '(-,n)' stands for none, 1, or more

[3] In the database field, the language that is used for data (schema) definition is called the DDL (Data Definition Language); the language used for data manipulation is called the DML (Data Manipulation Language). SQL is both the DDL and DML for current relational database systems.

```
STUDENT(STUDNAME, ADDRESS, EMAIL, TYPE)
FOLLOWSCLASS(STUDNAME, CLASS#)
COURSE(COURSE#, NAME, STARTDATE, ENDDATE, ROOM#)
ROOM(ROOM#, BUILDING, FLOOR)
```

Figure 1.4: Example relational schema

In the relational model, the main construct is the relation. A relation has a number of attributes and the meaning of the relation is entirely defined by the set of tuples that correspond to this relation. This means the meaning of a relation is not self-contained. The name of a relation has no formal meaning.

Tuples of a relation are identified by those attributes that have been said to characterize them.

In the relational model there are no explicit relationships between different entities. These relationships must be retrieved using queries on the database and some additional knowledge that is not contained in the relational model.

The Entity-Relationship model is often used for the modelling of Relational databases. However, in the translation for ER model to relational schema, a lot of information is lost. In ER modelling, entity types are explicitly specified and have a meaning, whereas these entity types only survive as meaningless names in the relational model. Furthermore, the relationships explicitly expressing in the ER model, are hidden in the relational model. These relationships can only be recovered using (external) *foreign key* constraints.

An example relational schema is presented in Figure 1.4. Underlined attributes make up the primary key, while *slanted* attributes indicate a foreign key.

In the example in Figure 1.4, we have modelled a student, consisting of a name (STUDNAME), an address, en email address and a type (e.g. 'Bachelor', 'Master' or 'PhD'), a class, consisting of a number (COURSE#), a name, a start date, an end date and a room number (ROOM#). A room has a room number (ROOM#), a building and a floor. The FOLLOWSCLASS relation has been introduced to enable the translation of an *n2m* relationship between student and class from the conceptual model to the relational schema. The way the relationship between STUDENT and COURSE is specified is through the specification of two attributes that are linked, through foreign key constraints, with the primary key attributes in the STUDENT and the COURSE relation.

Note that there is no indication of a relationship between the COURSE and the ROOM relation by just specifying the ROOM# attribute. An additional *foreign key constraint* needs to be specified, which states that the values for the ROOM# attribute in the CLASS relation should correspond to a value of the ROOM#[4] attribute in the ROOM relation.

Apparently, there are two ways of specifying relationships between relations in the relational model. One is through the specification of *foreign key constraints* (denoting a 1-to-m relationship) and the other is through the specification of an additional relation and a set of additional foreign key constraints.

[4]Note here that the similarity in the name of the ROOM# attribute in the CLASS and ROOM relations is incidental

1.2.3 Semantic data modelling

There is, however, a novel approach in database modelling, the semantic data modelling [ter Bekke, 1992] approach, which does have formal and real-world semantics. This modelling approach has been implemented in the Xplain database system [ter Bekke, 1992]. It has been shown that this approach solves some of the problems with SQL [ter Bekke, 1997a; Date, 1984]. The main advantage of this approach, in our opinion, is the combination of a sound semantic data model with an efficient implementation, by making a big distinction[5] between the conceptual data schema and the internal schema in the Xplain database management system. Xplain supports performative query execution and data retrieval, while retaining a sound conceptual model, with real-world semantics.

A data model that can potentially overcome this problem is the semantic database model [ter Bekke, 1992]. The semantic database model has a modelling language with powerful constructs, while retaining modelling simplicity. Three object types are distinguished in semantic data modelling:

Classification The domain[6] is described by capturing properties of relevant objects in a process called classification. The meaning of these properties is not defined by the instances occurring in the database, as in the relational model. The semantics of these properties is self-contained.

Aggregation An aggregation is the collection of a number of properties in a type, which is itself again a property.

Generalization When several types share similar attributes, these shared attributes can be combined in a new type, which is now the generalization for the other (specialized) types. Note here the similarity with generalization/specialization in Object-Orientation. The specialized types "inherit" all attributes from the general type.

These object types can be summarized using the concept of object relativity. By object relativity is meant, the description of an object in terms of the relationships with other objects. It is important to realize here that an object can play different roles in the different relationships. In one relationship it can be a generalization of a set of other objects, while in another relationship it might be a specialization of a more general object. An object can also be both a type and an instance. And also both an aggregation of other types and an attribute, aggregated in a different type.

An example semantic database model is depicted in Figure 1.5. The corresponding type definitions, which can readily be be used in the Xplain database management system [ter Bekke, 1992], is depicted in figure 1.6.

Figure 1.5 presents the so-called abstraction hierarchy. In the abstraction hierarchy, both the aggregation and the generalization relationships are shown. We see here that 'room' is aggregated in 'class' and both 'course' and 'student' are aggregated in 'attendance'. The generalization relationship shown in the figure is the generalization of the 'Msc-student' and 'PhD-student' types into the more general 'student' type. If we compare this diagram with the ER Diagram

[5] This in contrast with relational database systems, where the distinction between the conceptual and the internal schema is not very big.

[6] Also: relevant part of the real world or universe of discourse

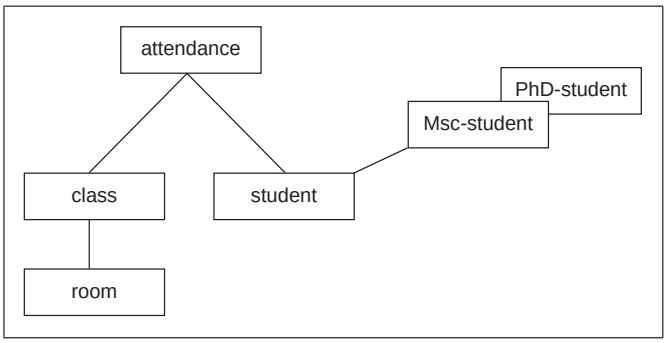

Figure 1.5: Example semantic database model

```
type student = name, address, email.
type Msc-student = [student] .
type PhD-student = [student] .
type course = name, start_date, end_date, room.
type room = room_number, building, floor.
type attendance = student, class.
```

Figure 1.6: Example semantic type definitions

in figure 1.3, we see that on the one hand, semantic database modelling is more expressive, because we expressed the generalization relationship between 'student' on the one side and 'Msc-student' and 'PhD-student' on the other. On the other hand, the semantic data model is simpler, because relationships do not need to be considered separately.

This is a drawback, however, of the semantic data model. The type 'attendance' in Figure 1.5 represents a relationship between the types 'student' and 'class'. There is unfortunately no other way to express this many-to-many relationship. This is where the ER model is more expressive, in that it is possible to express arbitrary cardinalities for relationships.

The type definitions in Figure 1.6 can be readily implemented in the Xplain database management system. However, we did leave out a number of definitions to make the figure more readable. Those definitions are the base definitions. A base definition defines a base type that is constructed from the built-in data types, such as integer, string, etc... A base type definition could, for example, state that the base type 'name' is a string with a length of 40 and that 'floor' is a positive integer.

Because a name uniquely identifies a type and types are self-contained, a mention of the name of a type in the definition of another type, means a reference to the other type. When the name of the other type is enclosed in square brackets, a generalization relationship is depicted. In the example, both 'Msc-student' and 'PhD-student' have [student] in their respective definitions, which means that they are both specializations of 'student'. The specification of 'room' in the 'course' type definition depicts the aggregation relationship between the two types, as shown in Figure 1.5.

In the semantic data model, the concept of *convertibility* plays an important role. Convertibility in type definitions means that a type (depicted by its *type*

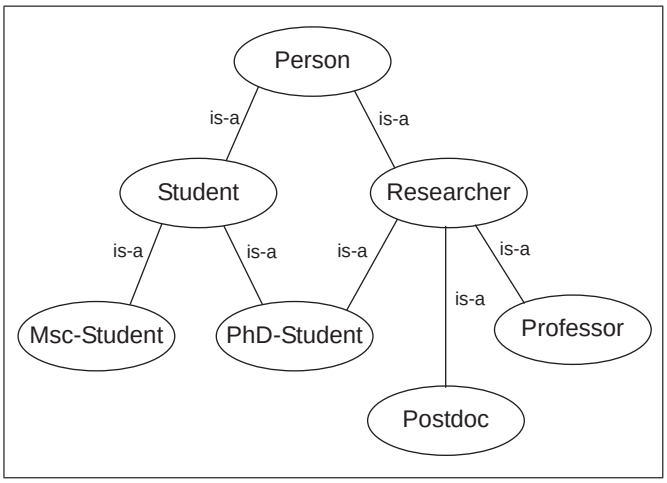

Figure 1.7: A simple example ontology

name) can only have one set of attributes and a set of attributes can only belong to one type. This means it is illegal to have two type definitions for the same type and it is illegal to have two types that have exactly the same attributes.

The convertibility principle for semantic data modelling is somewhat similar to the concepts of *necessary* and *sufficient* definitions and the way concept definitions are handled in Description Logics, as can be seen in the next section, where we introduce the Web Ontology Language OWL.

Just like for Entity-Relationship Diagrams, there exists a translation from the Semantic Data model to the Relational model (or rather the implementation language SQL, which is used by virtually all relational database systems) [de Boer and ter Bekke, 2001], in order to enable benefitting from the expressivity and conceptual simplicity of the conceptual modelling language. It turns out that the Semantic Data modelling language is easier to use than the Entity-Relationship model [ter Bekke, 1997b].

1.2.4 Introducing ontologies

Ontologies promise to provide a human and machine understandable model of a domain, where the domain is that part of the real world with which we are concerned. Ontologies facilitate *knowledge sharing and reuse* through their formal (machine-understandable) and real-world (human-understandable) semantics [Fensel, 2003]. By "semantics" we mean the intended *meaning* of the concepts. We illustrate the formal and the real-world semantics using the simple ontology presented in figure 1.7.

Our example ontology (Figure 1.7) consists of a simple formal taxonomy[7] with only *is-a* relationships between the concepts. A *Student* is a $Person$; a *Researcher* is a $Person$; an $Msc - Student$ is a *Student*, and so on.

The formal semantics in our example consists of strict inheritance. This strict inheritance means that every instance of the concept *Student* is also an

[7] A taxonomy is a hierarchy of concepts [Studer et al., 1998] where more general concepts are situated higher in the hierarchy.

instance of the concept *Person*; the *is-a* relationship is formalized.

The real-world semantics in our example consist of the usage of names that have a meaning for a human. For a computer the name of a particular concept doesn't matter, the concepts could be called c1, c2, etc. and a computer would still understand the ontology in the same way. The formal and the real-world semantics are connected in that every concept that has meaning in the real world, also has a formal, machine-understandable meaning. The concept *Student* in Figure 1.7 has a meaning for a human, but also for a computer. This connection between real-world and formal semantics does not exist in many popular database models, as we will see in the sections below. Especially the relational model does not attach any formal meaning to the name of a concept (or relation); the semantics of the semantic data model, however, are more similar to the semantics of typical ontology languages, in that here, just as in ontologies, real-world and formal semantics are interweaved.

If we translate Figure 1.7 to Description Logics [Baader et al., 2003] (described in more detail in [de Bruijn, 2003] and the section on OWL below), which is a popular class of languages for ontologies, we get the following description:

$$Student \sqsubseteq Person$$
$$Researcher \sqsubseteq Person$$
$$Msc - Student \sqsubseteq Student$$
$$PhD{-}Student \sqsubseteq Student \sqcap Researcher$$
$$Postdoc \sqsubseteq Researcher$$
$$Professor \sqsubseteq Researcher$$

We will see later on what these statements actually mean.

Below, we provide an introduction into the new Web Ontology Language standard OWL, which is the new ontology language for the Semantic Web (cf. 5).

1.2.5 The Web Ontology Language OWL

The World Wide Web Consortium [8](W3C) has set the standard for the ontology language on the Semantic Web, with the name of OWL [Bechhofer et al., 2003]. OWL has three dialects with the names of OWL Lite, OWL DL and OWL Full. Each lower (less expressive) dialect is both syntactically and semantically contained in the higher (more expressive) dialects, where OWL Lite is the least expressive and OWL Full is the most expressive dialect.

OWL is based on Description Logics (earlier called *terminological logics*) [Baader et al., 2003; Baader et al., 1991], which is a class of logic languages with a history in knowledge representation[9]. In the 1980s several early Description Logic-based Knowledge Representation systems were developed, such as CLASSIC [Borgida et al., 1989] and KL-ONE [Brachman and Schmolze, 1985].

In the beginning of the 2000s, the Web ontology language OIL (Ontology Inference Layer) [Fensel et al., 2001] was created, combining Description Logics, with its many efficient implementations of reasoners, and frame-based primitives, used to support the development and maintenance of ontologies, with

[8]http://www.w3.org/

[9]Note that ontologies come from the Artificial Intelligence research area of Knowledge Representation. An ontology is used to represent some knowledge which is held by some group of people.

XML and RDF to create an ontology language based on Web standards. In 2001, the DAML+OIL language [Horrocks and van Harmelen, 2001], a combination of the DAML-ONT [McGuinness et al., 2003] and the mentioned OIL efforts, was finalized and submitted to the W3C as the basis for the new Web Ontology Language (i.e. OWL).

Both OIL and DAML+OIL have a translation to the expressive $\mathcal{SHIQ}$ Description Logic language, for which efficient reasoning implementations exist (cf. [Horrocks, 2002]).

Description Logic Basics I will now try to explain Description Logics without going into too much detail.

There are three important concepts in Description Logics. These are concepts, roles and individuals. The major strength of Description Logics is its ability to classify concepts and individuals using descriptions of individuals and definitions of concepts. For example, if there exists a concept definition stating:

$Student \equiv Msc - Student \sqcup PhD - Student$

Which means that the concept *Student* encompasses all individuals in the concepts of $Msc - Student$ and $PhD - Student$. It also means, because of symmetricalness of the equivalence relationship ($\equiv$), that all individual for the concept *Student* are either contained in the concept $Msc - Student$ or in the concept $PhD - Student$ or in both[10]. We now know for every particular individual that is an instance of the concept $Msc - Student$ or an instance of the concept $PhD - Student$, that it is also an instance of the concept *Student*. We have, however, also excluded the possibility of having other students than Msc or PhD students. If we were to introduce a concept $Bsc - Student$, it would not be possible to group it under *Student*, unless we change the definition of *Student* to incorporate $Bsc - Student$.

The most important aspect of Description Logics, from a modeler's point of view, is that roles are specified independently from concepts. This allows for automatically determining the class of an individual based on the roles specified for that individual. This is conceptually a nice feature, but not always very clear for the human user, because humans tend to think in roles belonging to concepts and individuals specified directly as belonging to specific concepts.

For a more elaborate introductory example for Description Logics we refer the interested reader to section 3.2.2 in [de Bruijn, 2003] and for a complete introduction and elaboration of Description Logics we refer the interested reader to [Baader et al., 2003].

OWL Full combines RDF with the expressive $\mathcal{SHIQ}(\mathbf{D})$ (which is $\mathcal{SHIQ}$ extended with datatype support) Description Logic language in order to create an ontology language for the Semantic Web, which is (both syntactically and semantically) layered on top of RDF. OWL DL restricts OWL Full to that part, which is still (theoretically) decidable. OWL DL is equivalent to the $\mathcal{SHOIN}(\mathbf{D})$ Description Logic. Reasoning in OWL DL is still very hard; entailment (proving that one statement in OWL DL is a logical consequence of another) has a worst-case complexity of NExpTime (non-deterministic exponential time) [Horrocks et al., 2003].

[10] An individual can be both a Msc-Student and a PhD-Student, because we have not stated explicitly that this is not possible.

OWL Lite restricts OWL DL further to a language which has the expressivity of the $\mathcal{SHIF}(\mathbf{D})$ Description Logic. In the $\mathcal{SHIF}(\mathbf{D})$ language, satisfiability is computable in ExpTime (exponential time). And because entailment in OWL Lite can be reduced to satisfiability in $\mathcal{SHIF}(\mathbf{D})$ [Horrocks and Patel-Schneider, 2003], entailment in OWL Lite can be computed in ExpTime.

We will restrict ourselves in the remainder to the OWL DL variant, because, in the author's opinion, OWL Lite does not provide sufficient expressiveness for fully-fledged ontologies and OWL Full is not very useful as an ontology language, because it is not decidable, which is due to the layering on top of RDF(S).

1.2.6 Comparing ontology languages and conceptual database modelling languages

Most comparisons between ontology languages and database schemas compare ontologies with physical database schemas (e.g. [Fensel, 2003]), instead of conceptual database schemas.

[Fensel, 2003] mentions four major differences between database schemas and ontologies:

- Ontology language are both syntactically and semantically more expressive than common approaches for databases.
- The information that is described by an ontology can consist of semi-structured natural language texts and is not restricted to tabular data.
- An ontology **must** be a shared and consensual terminology, because it is used for information sharing and exchange. This aspect has to do with the purpose of an ontology and not so much the language that is used for data modelling. Ontologies are, from the start, created with information sharing in mind, while database schemas are usually created with a specific application in mind; information sharing is only secondary.
- An ontology provides a domain theory and not the structure of a data container.

We can see that not all of these differences hold for a comparison between ontologies and conceptual database schemas. The structure of a data container is specified in the internal schema of the database and not the (high-level) conceptual schema. A conceptual schema also provides a domain theory, which is, however, still limited to the application for which the database is used.

We will now describe the most important differences between conceptual database schemas (mostly geared towards the Semantic data model [ter Bekke, 1992]) and ontologies (assuming OWL DL as the ontology language):

Expressivity Ontology languages are more expressive than conceptual database languages. Classes can be created from existing classes, properties and individuals using equivalence, subsumption (i.e. subClassOf), intersections, unions, the oneOf construct and property restrictions. The Semantic data model uses just two types of relationships, namely aggregation and generalization.

Open world vs. Closed world Ontologies operate in an open world where not all knowledge is local and not all concepts and individuals are known. For ontologies on the Web the Open World Assumption (as in most logic languages) holds, which means that if a fact is not explicitly stated and it can not be derived from all known facts, it does not mean that it is not true. When the Closed World Assumption (as usual in database systems) is used, the absence of a fact implicates its negation (i.e. the fact is not true).

We can illustrate this open world vs. closed world assumption with an example application. Say, we want to display to the user a list of all students at a certain university. This is easy in a database system, where the closed world assumption holds. The application can just enumerate all values in the 'Students' table. Because of the closed world assumption, there are no students out there that we don't know about. Also, we know, for example, that all values in the 'Employees' table are not students if this has not been explicitly stated.

In a system where the open world assumptions holds, it is in fact not possible to guarantee that all students are enumerated. It is possible that there are students out there that we don't know about, but that do study at our university. Furthermore, members of other concepts, for example of the concept 'Employee', might also be students if the contrary has not been explicitly stated.

The closed world assumption has advantages in the sense that it is far easier to reason with. Many more conclusions can be drawn when using the closed world assumption, namely the negation (the opposite) of all not explicitly stated facts.

Unique name assumption In the semantic data model, the unique name assumption holds. There can not be two *type names*, which denote the same type, where the meaning of the type is defined by its attributes (i.e. relationships with other types). There can not be two individuals with a different identification that represent the same object in the real-world.

In OWL, the unique name assumption does not hold. The same concept or individual can be described by different concepts names or different individuals names, respectively. Equivalence between concepts or individuals can be inferred from their descriptions.

More specifically, one does not know whether two different names refer to the *same* object, or whether they refer to *different* objects. It is often possible to infer (conclude) that two names refer to different objects using the descriptions referred to by the objects or by explicit inequality statements of the two names.

We can illustrate this with an example. Say, the individuals 'jane' and 'joe' are instances of the concept 'Employee'. Now, jane and joe might have different names, but we do not have enough information to conclude that they are different; for all we know 'jane' and 'joe' refer to the same actual employee in the real world. If we want to be sure that jane and joe are interpreted to be different, we should explicitly state their non-equivalence ($jane \neq joe$).

```
<rdf:RDF>
  <owl:DatatypeProperty rdf:about="#birthdate">
    <rdfs:domain>
      <owl:Class rdf:about="#student"/>
    </rdfs:domain>
    <rdfs:range rdf:resource="&xsd;date"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:about="#name">
    <rdfs:domain>
      <owl:Class rdf:about="#student"/>
    </rdfs:domain>
    <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:about="#type">
    <rdfs:domain>
      <owl:Class rdf:about="#student"/>
    </rdfs:domain>
    <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:about="#email">
    <rdfs:domain>
      <owl:Class rdf:about="#student"/>
    </rdfs:domain>
    <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DatatypeProperty>
</rdf:RDF>
```

Figure 1.8: Example RDF/XML serialization of an OWL ontology

```
Ontology(

 Class(a:student)

 DataProperty(a:birthdate
  domain(a:student)
  range(xsd:date))
 DataProperty(a:email
  domain(a:student)
  range(xsd:string))
 DataProperty(a:name
  domain(a:student)
  range(xsd:string))
 DataProperty(a:type
  domain(a:student)
  range(xsd:string))
)
```

Figure 1.9: Example abstract syntax serialization of an OWL ontology

Intended usage The intended use of conceptual database schemas is to model single databases for single applications, where ontologies are intended to be used for the specification of domain theories that are shared by groups of people or organizations on the Semantic Web.

Simplicity The layering of OWL on top of the other Semantic Web languages XML and RDF may provide a better exchangeability of OWL ontologies[11], but this does not make the language very human-readable, because of the verbose nature of the RDF/XML serialization. There is an abstract syntax for OWL [Patel-Schneider et al., 2003], which provides a more human-understandable syntax than the RDF/XML exchange syntax (cf. Figures 1.8 and 1.9). However, because of the Description Logics basis, this abstract syntax is still hard to read because of the separation of the class and the property definitions. Furthermore, OWL does not support the modeler of the ontology because of the lack of orthogonality in its modelling constructs. Concepts can be modelled in many different ways and are often not even explicitly modelled, but inferred by the Description Logic reasoner, which is not very clear to the modeler. Therefore, modelling constructs in OWL lack orthogonality. The Semantic data model does provide orthogonality in its modelling constructs [ter Bekke, 1992].

In this comparison we have mainly looked at the Semantic data model for database modelling and the OWL DL language for the specification of ontologies. However, these languages are both currently not widely in use, for different reasons[12].

[11] This, however, remains to be seen

[12] The Relational model is still dominant in the databases area, because of the availability

We will now look into ways to consolidate existing data sources and the schemas describing these sources using ontology technology. We will describe generic approaches to the information integration problem described above and in the subsequent chapters we will look into several specific solutions to the problem both within and between (parts of) enterprises.

1.3 Approaches in Semantic Information Integration

We identify two major paradigms in information integration: (1) merging data models into a central model and (2) aligning and mapping models. In the ontology engineering community these approaches are known as Ontology Merging and Ontology Aligning.

[Noy and Musen, 1999] clarify the difference between ontology merging and ontology aligning. When merging two ontologies, a single coherent ontology is created that is a merged version of the two original ontologies. When aligning two ontologies, the two original ontologies persist, with a number of links established between them, allowing the aligned ontologies to reuse information from one another. Therefore, the alignment of ontologies is usually part of the ontology merging process.

Solutions can be further classified along two dimensions: a run-time and a design-time dimension. In the run-time, or user-centered, dimension we distinguish two approaches: the (1) local model and the (2) global model approach. The difference between these two approaches is whether, in interactions with the system, the user can use his/her own local data model, or whether the user needs to conform to a global model when interacting with the system:

- In the *local model*, or local ontology, approach the user is represented by an agent in the system and this agent presents the user with its own local data model. The agent performs the translation between the user's local model and either the global model or other local models in order to allow interaction with multiple data sources in the system. And example of the local model approach is the KRAFT project [Preece et al., 2001].
- In the *global model*, or global ontology, approach the user will view the system through the global data model using a mediator, which is "a system that supports an integrated view over multiple information sources" [Hull, 1997]. Note that in the local model approach, a user agent will in most cases also contact a mediator in order to allow inter-operation with the system, which contains multiple information sources. An example is the approach taken in the COG project, which is described in detail in the next section.

The run-time dimension concerns with the way the user views the data in the system during operation. The design-time dimension concerns with the way

of many commercial databases management systems implementing this model, impeding the transition to a better data model. The OWL language is still very new and at the time of write not even yet an official standard, although this is expected to happen very soon.

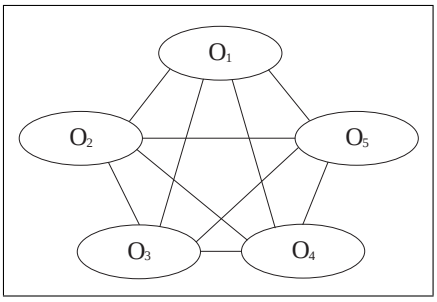

Figure 1.10: An example of one-to-one mapping

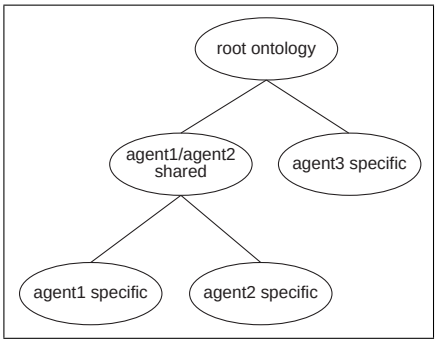

Figure 1.11: An example of ontology clustering

the models of the disparate data sources are integrated. We distinguish (1) one-to-one mapping, (2) using a single-shared ontology and (3) ontology clustering [Ding and Foo, 2002]:

- *One-to-one mapping* of ontologies. Mappings are created between pairs of ontologies. Problem with this approach arise when many such mappings need to be created, which is often the case in organizations where many different applications are in use. The complexity of the ontology mapping for the one-to-one approach is $O(n^2)$ where n is the number of ontologies. An example of the one-to-one approach is OBSERVER [Mena et al., 2000]. Figure 1.10 illustrates one-to-one mapping of ontologies. There exists a mapping between every pair of ontologies. In the worst case, these mappings are only one-way. This means that a single mapping can only translate from one model to another, not the other way around.

- Using a *single-shared ontology*. Drawbacks of using a single-shared ontology are similar to those of using any standard [Visser and Cui, 1998]. For example, it is hard to reach a consensus on a standard shared by many people (it is always a lengthy process), who use different terminologies for the same domain and a standard impedes changes in an organization (because evolution of standards suffers from the same problems as the development of standards). Examples of the single-shared ontology approach are MOMIS [Bergamaschi et al., 2001] (see also section 4.1.1) and the Semantic Information Management [Schreiber, 2003], which is the methodology used in the COG project and is described in detail in chapter 2.

  Within the paradigm of single-shared ontology mapping, we distinguish two forms:

  - *Removing* the old local data models. All applications use the new global data model. A drawback of this approach is that applications depending on the local data models will break and have to be adapted to the now global model. Another drawback is the fact that groups in the organization can no longer maintain their own terminology; everybody will have to submit to the new global model [Uschold, 2000].

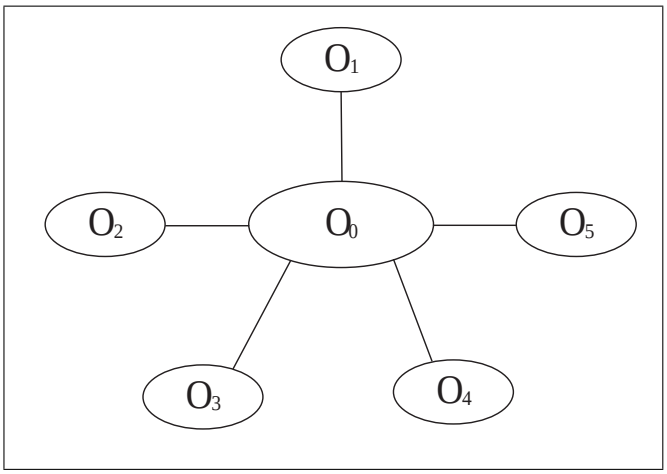

Figure 1.12: An example of ontology mapping using a single-shared ontology

  - Keeping the local data models and creating a *mapping* to the new global data model. Local models can remain in place; applications will not break because of the new global model. A drawback of this approach is that still old (possibly not so good) data models remain and mappings need to be maintained. They need to be updated with every update of the local model and with every update of the global model.

- *Ontology clustering* based on the similarity of concepts known to different agents [Visser and Tamma, 1999]. The ontology clusters are organized in a hierarchical fashion, where the root node is the most general cluster. A lower level in the hierarchy corresponds to a more agent-specific, less abstract representation of the domain. An example of ontology clustering is illustrated in figure 1.11. The agent ontologies are typically mapped to leaves in the tree.

We will classify the tools and methods presented in this thesis using these dimensions.

This thesis is organized as follows: in chapter 2 we introduce information integration inside the enterprise in the context of the COG project and we present the solutions developed in the project. In chapter 3, we show how the semantic information integration architecture developed in the COG project can be used to aid retrieving information from sources throughout the enterprise. Then, in chapter 4, we provide a comparison of the Semantic Information Integration solution in the COG project with other state-of-the-art semantic information integration projects and tools. Chapter 5 broadens the scope of Semantic Information Integration to inter-organizational inter-operation on the Semantic Web and present work done in that area in the Esperonto project. Finally, in chapter 6, we provide conclusions and an outlook.

Chapter 2

Semantic Information Integration in the Enterprise

This chapter[1] is partly based on a White Paper to be published as a part of the COG project at http://www.cogproject.org/, with the title 'Semantic Information Integration in the COG project' [de Bruijn et al., 2003a].

In the Corporate Ontology Grid (COG) project we have created an information architecture to support locating and accessing data residing in heterogeneous databases, and other, less structured data sources, throughout an industrial organization. In order to realize this architecture we had to integrate the heterogeneous data sources and present the user with a unified view of the disparate data schemas to enable browsing and querying of data located throughout the organization.

We chose to create a global schema, integrating all source schemas, that acts as a mediator[2] between the various data sources. When the mappings between the source schemas and the global schema have been created, the user can browse the global schema and in this way discover what information is present in the organization and where the information is located. The user can also issue queries to the mediator that are automatically translated to the respective platforms and schemas of the data sources, where they are then executed. Using the mappings from the various source schemas to the global schema, it is also possible to automatically derive transformations of instances between different schemas.

We used a semantic approach to the integration problem instead of abstracting a global database schema from the source schemas. The meaning of the data is captured in a central ontology and the data in the sources is given meaning by creating mappings between the sources and the ontology. The central ontology is an integrated *virtual* view (as opposed to a *materialized* view [Hull, 1997])

[1] Some materials presented in this chapter are the copyright of Unicorn Solutions, Inc. and are used with permission.

[2] For more information about the role of mediators in Information Systems, please refer to [Wiederhold, 1992] and [Wiederhold and Genesereth, 1997]

of the information present in the enterprise. This means the original platforms and data schemas remain and are not replaced by the global schemas and the applications using these data sources do not need to be changed when creating the central ontology.

In the COG project we use data sources from different platforms and with different data schemas. The differences in these data schemas imply the necessity of not only creating a simple translation from one platform to another, but also a semantic mapping, linking the entities in the disparate schemas based on the correspondence in their meanings. When the central ontology has been created, it is possible to create mappings between the source schemas and the ontology, thereby semantically linking the terms in the sources to each other and providing a meaning for the data residing in each source related to the meaning of the data residing in other sources.

In the COG project we integrate a number of existing heterogeneous data sources, using different platforms and (syntactically and semantically) different data schemas, from the automotive industry by creating a central ontology integrating these data sources. We created mappings between the source schemas and the ontology, thereby creating an integrated unified global virtual view [Hull, 1997] of the information present in the disparate data sources throughout the enterprise and enabling the querying of the disparate data sources via the central ontology.

The data sources to be used for the COG project are provided by CRF[3] (Centro Ricerche FIAT). There were several relational (RDBMS) database sources, XML sources, spreadsheet documents, and PDF documents. All these sources had to be integrated in the information architecture.

For the implementation of the information architecture, the methodology described in Unicorn's[4] Semantic Information Management (SIM) [Schreiber, 2003] was used together with the Unicorn Workbench[5] [Unicorn, 2003b] tool, which supports the SIM Methodology. We first give an introduction into the Semantic Information Management and the SIM Methodology, then we describe the Unicorn Workbench, and finally we describe our experiences using the Semantic Information Management and the Unicorn Workbench in the COG project.

A summary of all the Unicorn (and other) terminology can be found in the glossary of this thesis.

2.1 The Semantic Information Management

As pointed out in the introduction to this thesis, there are three main paradigms in the data source integration problem. The first one is the one-to-one mapping paradigm (cf. Figure 1.10) where ad-hoc mappings between individual sources are created; the second main approach is using a single-shared ontology for all the applications (cf. Figure 1.11); the third approach is ontology clustering (see figure 1.11 and the description in section 1.3).

[Uschold, 2000] pointed out three possibilities for using a global ontology together with local ontologies. Either a global ontology does not exist and only

[3] http://www.crf.it/

[4] http://www.unicorn.com/

[5] The Unicorn Workbench is part of the Unicorn System tool suite

local ontologies are used, or a global ontology exists, either with or without local ontologies. When only local ontologies exist, this corresponds to the one-to-one paradigm. When only a global ontology exists, this corresponds to the single-shared ontology paradigm. When a global ontology exists alongside local ontologies, we have a mix between the one-to-one and the single-shared ontology paradigms; there is a shared ontology and there is a mapping between the central ontology and each local ontology[6]. The approach taken in the Semantic Information Management (SIM) is the latter[7].

The aim of the Semantic Information Management is to provide enterprises with insight into the information residing in different sources in different formats with different schemas across the enterprise (this is also known as the Enterprise Data Problem). The SIM aims to provide a solution to this problem by creating a central ontology (also called the 'Information Model') and mapping the individual source schemas to this central ontology, thereby creating a global view of all data residing in the organization along with an insight into the location of the data.

The SIM approach to information integration consists of three stages. First, the metadata of the existing data sources is to be collected. Then, using this metadata, a central ontology is created capturing the meaning of the data present in these data sources. Finally, the disparate data schemas are mapped to the ontology in a process of rationalization, in order to give semantics to the data residing in the various sources.

The Semantic Information Management is supported by a tool created by Unicorn, called the "Unicorn Workbench" (hereafter referred to simply as *Unicorn*). Unicorn was created to support all phases in the SIM methodology.

The SIM describes the knowledge model for the ontologies in Unicorn as consisting of five layers, namely:

1. *Organizational layer.* The Information Model is divided into a number of subject areas, called *packages*. Each package reflects a different part of the business. The different packages can be viewed as different, interrelated sub-ontologies. Packages can be organized in a hierarchy, even though this is conceptually just a way of grouping these packages, which means that there is no formal relationship between a package and its sub-package.

2. *Entity layer.* The *classes* (concepts) are captured in a class-hierarchy. The classes are derived from the composites present in the various data sources.

3. *Property/attribute layer.* In this layer the *properties* of the classes are captured. These properties constitute the relationships between the classes, besides the is-a relationship, already captured in the entity layer. A property in Unicorn is a second-class citizen, defined as part of a class, as in frame-based languages such as F-Logic [Kifer et al., 1995], other than the class-independent property definitions in description based ontology languages, such as RDF(S) [Lassila and Swick, 1999; Brickley and Guha, 2003] and OWL [Bechhofer et al., 2003].

[6]Note that we can use ontology and data model interchangeably here

[7]In SIM, a central ontology ('Information Model' in Unicorn terminology) is created along with mappings to individual data sources ('external assets' in Unicorn terminology)

4. *Business rule layer. Business rules* related to the values of the properties are created. Relations between properties as well as value restrictions for properties (e.g. using enumerated values and lookup tables) are established. The language used for conversions of values in Business Rules is a subset of Python [Unicorn, 2003b] constrained by a strong typing-system based on the fundamental data types.

5. *Descriptor layer.* Besides the formal description of elements in the ontology, as is done in the top four layers, there exist informal descriptions of the concepts in the ontology, in order to enhance human understanding. Each concept (such as classes, properties, etc...) in the ontology has a number of (customizable) descriptors that can be used for this purpose.

As pointed out in [Visser and Cui, 1998] and [Uschold, 2000], problems can arise in maintaining or developing a single-shared ontology. [Visser and Cui, 1998] identified four drawbacks of using a single-shared (or "standard") ontology[8]:

- A standard ontology is a 'heavy vehicle' because it must encompass all the terminology used in the organization; it must cover every possible communication need.
- Both defining and maintaining a standard ontology are hard tasks, because of the need of consensus in a large group of designers, users, managers, etc...
- Using a standard ontology impedes heterogeneity among applications. Current software applications using their own local schema will have to be rewritten in order to use the standard.
- Standards hinder changes in the communication in the organization, because of their inflexibility (i.e. changing a standard requires a considerable amount of time and effort).

In the Semantic Information Management approach, such a single-shared ontology is created to unify the disparate data sources, so one would expect these problems to occur in solutions based on SIM. However, in the SIM approach, the single-shared ontology is not used in the traditional sense, where the source schemas are deleted and the applications are required to use the new global schema. Instead, the created global schema is virtual (cf. [Hull, 1997]), allowing the existing schemas and application to remain unchanged, with mappings between the original schemas and the new global schema.

Using a virtual global schema certainly allows heterogeneity among applications, because local source schemas remain. Also, flexibility is higher, because local schemas can change independently; the global ontology does not need to be changed[9]. However, one must take care in maintaining the mappings between the local and the global schema when updating the local schema. The

[8] [Visser and Cui, 1998] identified problems for defining ontology standards for different communications layers; we have narrowed the drawbacks to the terminology, or ontology, layer

[9] Note here that this holds only for minor changes for which the global ontology does not need to be updated; when, after a change, a local schema can no longer be mapped to the global schema, problems as mentioned in [Visser and Cui, 1998] will occur

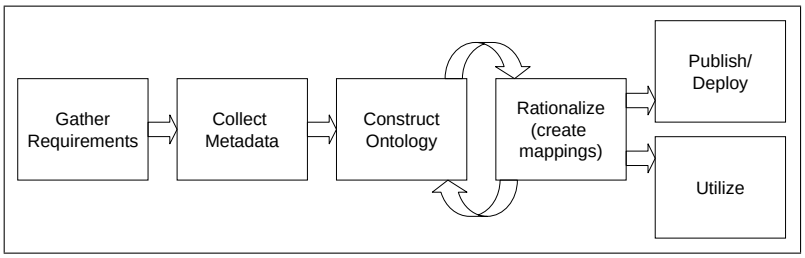

Figure 2.1: The Semantic Information Management Methodology

problems that remain are the size of the standard ontology and the problems in creating and maintaining global ontologies. We do not see a solution to the former problem in SIM; however, the latter problem can be partly solved in the *organizational layer*.

The organizational layer in SIM consists of packages that are used to group, amongst other things, classes. Different groups in the organization can be responsible for the construction and maintenance of different packages, where each package contains the terminology for the group. It is possible to define owners and permitted editors for packages in the Unicorn Workbench. In this way, different organizational units can maintain their own terminologies, which would require less consensus among different groups and would speed up the development and maintenance tasks. One should, however, keep in mind that these different terminologies need to be interrelated within Unicorn and one should be aware that each added package adds to the complexity of the overall Information Model.

2.2 The Methodology

In the COG project, we followed the Semantic Information Management (SIM) Methodology [Schreiber, 2003] for the creation of the ontology and the mapping of the disparate data sources. The SIM Methodology (see Figure 2.1) consists of six steps:

1. *Gather requirements.* Requirements for the information architecture are collected and the scope of the project is established.

2. *Collect Metadata.* All data assets relevant to the project are catalogued and the metadata (i.e. data schemas and existing conceptual models) are imported into the Unicorn Workbench.

3. *Construct Information Model.* Using the imported metadata, the ontology is created through a process of reverse engineering and/or manual identification of classes, properties, and business rules in the source schemas.

4. *Rationalize.* In the rationalization phase, the mappings between the data schemas and the ontology are created. If the model needs to be refined, there will be an iteration step back to phase three. In general, when creating mappings from the composites in the external assets to the ontology,

missing classes, properties, and business rules are discovered, which necessitates many iterations between the phases three and four to complete the model and the mappings.

5. *Publish/Deploy.* The Information Model, along with the mappings, is published to relevant stakeholders and the information model along with the transformations between the data sources is deployed to the runtime architecture.

6. *Utilize.* Processes need to be created to ensure maintenance of the architecture. Changes in the data sources need to be reflected in the ontology, mappings need to be updated according to these changes, etc ...

The implementation of phases two up to and including six of the methodology are facilitated by the Unicorn Workbench tool.

This chapter focusses on the third and fourth phase of the methodology, the support by the Unicorn tool, and our experiences with the methodology and the tool in the COG project. We first describe the Unicorn Workbench tool and then our experiences in the COG project.

2.3 The Unicorn Workbench

The Unicorn Workbench is a java-based tool, created by Unicorn, built to support the Unicorn Semantic Information Management [Schreiber, 2003] and to enable SIM implementations in enterprises. All phases (except the first) in the SIM Methodology are to some extent supported by the Unicorn tool.

The Unicorn Workbench is continuously being developed. Even in the course of the COG project, many new requirements were identified and implemented in the tool.

The basic concept in Unicorn is the Unicorn Project, which consists of the Information Model, the schemas belonging to the external assets (the data sources), the transformations, the queries, and the descriptors for all these concepts.

The architecture of Unicorn consists of two main layers (see figure 2.2), namely:

- The *External Assets layer* contains the mappings to all the (disparate) data sources. All kinds of data schemas can be imported into a Unicorn project, as long as there is a parser for it. Current supported formats are relational database schemas, XML schemas, as well as COBOL Copybooks. New parsers can be written using the Asset API. After importing the data schema, the user only needs to create the mappings between the concepts in the data source and the concepts in the ontology in order to complete the wrapper for the data source.

- The *Model Layer* contains the ontology (also called *Information Model*). The ontology contains all the packages, classes, properties, and business rules for describing the meaning of the data residing in the external assets.

All concepts in these two layers are documented in human-readable form using descriptors. For each different concepts (e.g. project, class, transformation, external asset, etc ...), there is a descriptor prescribing which concept

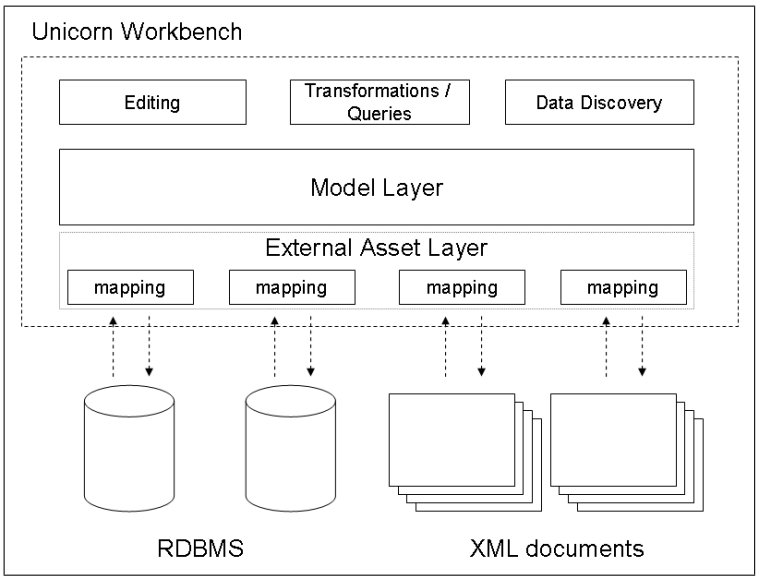

Figure 2.2: Semantic Information Management

attributes are available for documentation. There are some default descriptor files distributed with Unicorn, but they can be customized on a per-project basis. A concept attribute heavily used in the COG project is the 'synonym' attribute, used for several concepts (e.g. classes, properties). Italian translations for English-language terms, which were used in the original data sources and applications provided by CRF, are stored in these attributes.

The Model layer describes the meaning of the data and the External Asset layer describes the location of the data. In order to make the ontology active, the Unicorn Workbench provides three functions for the user. An editing function, used to create and maintain the ontology and the mappings to the different data sources. The second function is the data discovery function, which can be employed by the user to discover the location of data, residing in the disparate data sources, using the ontological model in Unicorn. Finally, there is a transformation and querying function with which the user can create transformations of instances between different data sources and issue queries against the ontology. The queries are syntactically translated to the query language of the data source and semantically translated (i.e. the query is automatically rephrased using terms from the external asset) to be used with the data schema of the source.

When mappings from two different data sources to the ontological model are in place, an instance transformation from one schema to the other can be generated by Unicorn. The transformation is generated in the form of a so-called Transformation Planner. This Transformation Planner is an XML document describing which composites and atoms from the source schema are to be mapped to which composites and atoms from the target schema. This XML document can be used to develop the actual transformation. In the workbench there is already support for generating SQL transformation scripts (for relational databases) and XSLT documents (for the transformation of XML documents).

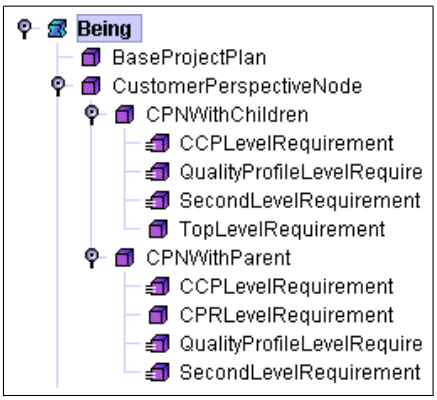

Figure 2.3: A part of the COG class hierarchy

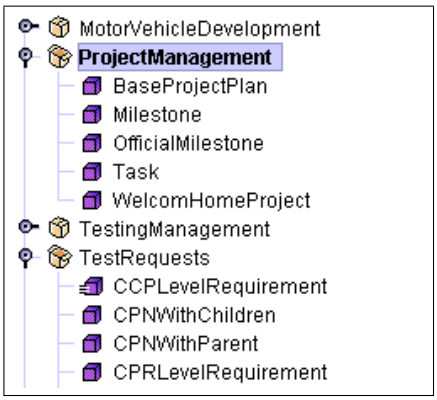

Figure 2.4: Classes organized in packages

Besides these external assets, other Unicorn ontologies and ERWin[10] models can be directly imported into the Model Layer of the project (the ontology). In this way, existing logical data models and component ontologies can be leveraged in the project. Using component ontologies decreases the development effort of the project and allows to leverage proven models.

2.3.1 Building an ontology using Unicorn

In the Unicorn Workbench, an ontology can be created in three ways: by reverse engineering the schema of an external data asset, by importing previously created (off-the-shelf) ontologies, or by building the ontology from scratch. The main concepts in the ontology are the packages, classes, the properties, and the business rules. Classes are organized in a class-hierarchy (taxonomy), supporting multiple inheritance (which means that a class can have two or more parent classes). Besides the hierarchical organization, classes can also be organized in packages, with each package representing a certain subject-area or business functions (e.g. project management, vehicle development, etc). These packages can also be organized in a hierarchy, but no inheritance is supported here and a package can only be grouped under one other package.

Classes in an individual package are required to have unique names, but across packages uniqueness in class-names is not required (this is done in order to make packages more independent), so it is possible for different classes to have identical names. This problem of duplicate names across packages is solved by prefixing the package name and a '.' to the class name. Furthermore, properties within a single class are required to have unique names. However, because of inheritance it is still possible to have two properties with the same name in a class. These properties are distinguished using to name of the originating class.

In Unicorn, classes can be viewed in two ways. One can view the class hierarchy (see Figure 2.3 for an example) or view the classes organized in packages (see Figure 2.4 for an example).

[10] An Entity-Relationship Diagram editor, see http://www3.ca.com/Solutions/Product.asp?ID=260

If some of the source schemas are designed well, it makes sense to use these as a basis for the ontology, because an existing data schema in an organization already represents a certain amount of consensus about knowledge in the domain [Meersman, 2001]. But even when such well-designed schemas are available to the ontology designer, the domain experts and end-users should still be involved with the engineering process in order to validate the ontology and to clarify the source schemas and the knowledge in the enterprise.

Relationships between classes other than the is-a relationship (the sub-class relationship expressed in the class-hierarchy) are created using properties. The type of a property can either be a class or a cross-product of two or more distinct classes in the ontology. Other constraints on the type of a property are cardinality constraints. The minimum cardinality of a property can be either 0 or 1. The maximum cardinality can either be one ('1') or one or more ('∗'). When the maximum cardinality is set to '∗', one can specify whether the values are ordered and whether the values can repeat.

When setting the maximum cardinality to one or more ('∗'), a complex class is used as the type of the property. There are four kinds of complex classes, based on whether the values for the property need to be ordered and whether the values can repeat. These four kinds are:

- *List.* The values are ordered and repetition is permitted.
- *Bag.* The values are not ordered and repetition is permitted.
- *Sequence.* The values are ordered and repetition is not permitted.
- *Set.* The values are not ordered and repetition is not permitted.

When the type of a property needs to be of a data type (e.g. integer or string), the user can use the classes provided in the `fundamental` and the `fundamental.time` packages. These classes representing the fundamental data types have properties for different representations of the value, as well as an identity property with the name of the class itself, which stands for an abstract representation of the data value itself. The `Integer` class, for example, has a `stringRepresentation` property with the class `String` as range. Inversely, the `String` class has an `integerRepresentation` property for representing the string value as an integer. This system enables the mapping of properties with different data types that represent the same concept (e.g. a number could be represented in one data source by an integer and in another by a string data type).

The values of properties can be restricted using *business rules*. There are six kinds of business rules:

- *Conversion* (transformation) between properties. Conversion formulae can be constructed using an expression language based on Python.
- *Equivalence.* When two properties are synonyms, this can be expressed with such an equivalence rule. It is most common to express this synonymity between a direct and an indirect property (i.e. property of a related class) of a particular class.
- *Enumerated values.* A complete list of possible values for the property is specified. An instance cannot have a value for this property that is not on the list.

- *Lookup table.* A lookup table relates the value of a property to the value of (one or more) other properties.
- *Uniqueness.* The value for one property or values for a combination of properties in one class can be said to be unique.
- *Type restriction.* In a subclass, the type of a property inherited from the superclass can be restricted to a subclass of the original type.

Business rules can be created for properties originating from superclasses. It is, however, not possible to edit property definitions of inherited properties, because they are second-class citizens belonging to a specific class. By defining business rules independent of properties, it is still possible to enforce constraints in subclasses that do not need to be enforced in superclasses. Note that business rules are inherited.

Conversion, enumerated values, and lookup table business rules can only be specified for properties with a fundamental data type. Equivalence, uniqueness and type restrictions can be used for properties with an arbitrary type. It must be noted here, however, that business rules can also be applied to indirect properties. This means that if a certain direct property has a non-fundamental type, but that non-fundamental class does have a (in)direct property with a fundamental type, conversion, enumerated values, and lookup tables business rules can still be used for that property.

Whenever a user wants to edit or delete an element that other elements depend on, Unicorn will perform an impact analysis to identify any issues (i.e. elements that have been rendered invalid by the action) caused by the action and inform the user what elements will be affected by the action and ask the user for confirmation. If the user still wants to perform the action, the impacted elements are marked as invalid and listed in a central location. The user is not allowed to use invalid elements in, for example, property and business rule definitions.

2.3.2 Mapping data schemas to the central ontology

During the metadata collection phase, a number of data schemas have been imported in the Unicorn project. These schemas have been used in the construction phase to aid in constructing the ontology. These source schemas now have to be mapped to the ontology in order to give meaning to the data residing in these sources and to enable locating data in the disparate sources and to enable data transformation between the disparate sources and issuing queries to the sources. Just like in the construction phase, it is very important to involve the domain experts in the mapping (rationalization) phase.

During the rationalization phase the user is aided by the Unicorn Workbench in creating the mappings between the data assets and the ontology. It is possible to either view the mappings from the viewpoint of the data assets and in this way determine for each type and property to which class/property it should be mapped and to create the mapping. Another possibility is to use the Data Discovery feature to drill down the class hierarchy to find out which mappings currently exist for the classes in the ontology. If a class is identified that requires further mapping, the designer can switch back to the view of the desired data asset and create the mappings.

If during the mapping phase, the designer discovers missing classes, properties, or business rules, the designer iterates back to the construction phase (phase three) to add the necessary concepts to the model. It is our experience from the COG project that especially missing properties and business rules are discovered in the rationalization phase and not so much missing classes.

Not necessarily all data schemas have to be mapped manually. If a data asset has been automatically reverse engineered during the construction phase, the mappings between the ontology and the external asset have already been automatically created. Also when a single entity in a source schema is reverse engineered into a class in the ontology, the mapping will have been created automatically.

Mappings are created in two stages. First the *Coarse Mapping* is created, linking composites in the data sources (e.g. tables, complex types) to classes in the ontology. Then the *Detailed Mapping* is created, linking atoms from the source schemas to the properties in the ontology.

Two different kinds of groups can be created in the mapping process. *Subject groups* are used to group composite types in the external asset to facilitate the mapping process for the user (the user selects a subject group and is shown only the composite types belonging to that group). The *instance mapping groups* are an integral part of the mapping itself. An instance mapping group represents a group of instances to be mapped to a certain class in the ontology. If only one instance group is defined for a composite, all instances of that composite will be mapped to the target class.

Conditions can be specified on the values of the atoms of the composite in the source schema. This way certain groups of instances can be mapped to different classes in the ontology. These conditional instance mapping groups are, however, not required to be mutually exclusive. Conditional mapping can not only be applied to classes (global conditional mapping), but also to individual properties (local conditional mapping), as mappings are eventually created on the property level (see the property layer in 2.1), during the detailed mapping. The other way around is also possible: different composites can be mapped to the same class.

The mappings in the detailed mapping stage are usually made between atoms and *direct* properties. A direct property is a regular property of the concerning class. It is however also possible to create mappings to *indirect properties*. An indirect property is not a property of the concerning class, but a direct or an indirect property of a class, which is the type of a direct property of the class (i.e. an indirect property is a property of a related class). It is therefore possible to map to an indirect property at an arbitrary depth.

When instances of different composite types in the source schema need to be mapped to a single class in the ontology, it is possible to create a mapping view. Such a mapping view can contain joins over different composites within one external asset. A join can be defined over the composites using an existing or an (in Unicorn created) "implicit" foreign key.

Foreign keys in the database[11] or "implicit" foreign keys are used either

[11] A note must be made here about these foreign keys and database mapping. Because not only databases can be mapping to the Information Model, but also XML documents, Cobol copy books, and others. The database primitives, such as 'foreign key', might not be appropriate for these sources; these difference are distinguished in the Unicorn Workbench. In section 2.4.2 we see an example of differences between internal representations of databases

to indicate a simple relationship with another class or to indicate inheritance. In the former case, the foreign key can be mapped to a property in the class that references the target class that represents the target composite type of the foreign key. In the latter case, the foreign key is mapped directly to the inheritance relationship. This latter case applies when the extension of the composite (i.e. the set of instances described by the composite) is a subset of the extension of another composite and this relationship is made explicit in the database using a foreign key.

When mapping external assets to the central ontology, it is possible to use so-called subtype mapping. An atom in a composite in the external asset can be mapped to a subclass of the type of the property in the ontology. This is of course a valid mapping, because a subclass of the original type is also a valid type for the property.

2.4 Ontology construction and rationalization in the COG project

In the COG project we used external assets provided by CRF (Centro Ricerche Fiat). The source schemas are taken from real-life data sources currently in use by CRF and sources to be implemented at CRF in the future. The goal of the project is to implement a single integrated (semantic) information architecture for the various information sources provided by CRF in order to show the applicability of using ontologies for information integration in industry. The sources include relational databases, XML data sources and PDF and spreadsheet documents. These PDF documents are accessed using the LiveLink document management system, which in turn has an XML interface. For the integration of the Excel spreadsheet documents, a special parser was written using the Asset API.

In the data assets, two different languages are currently in use, namely Italian and English. This was an opportunity to show how a central ontology can help bridge the language gap. The ontology is created using English-language terms, but it should be usable by people only familiar with the Italian-language terminology. To overcome this multi-lingual problem, Italian synonyms are used in the class and property descriptors in the ontology. The use of these synonym descriptors enables searching for the Italian terms and finding the related (English named) concepts in the ontology.

Because of the Italian terms used in a number of the source schemas, the reverse-engineering functionality of Unicorn could not be used. The Information Model had to be constructed manually on the basis of input from domain experts and afterwards the Italian-named atoms and composites were mapped to the (English named) classes and properties in the ontology in order to give them meaning.

2.4.1 The Information Integration Problem in COG

There are five main data sources (see also the COG Architecture in figure 3.4) provided by CRF to be integrated in the COG project. These sources consist of

and XML sources.

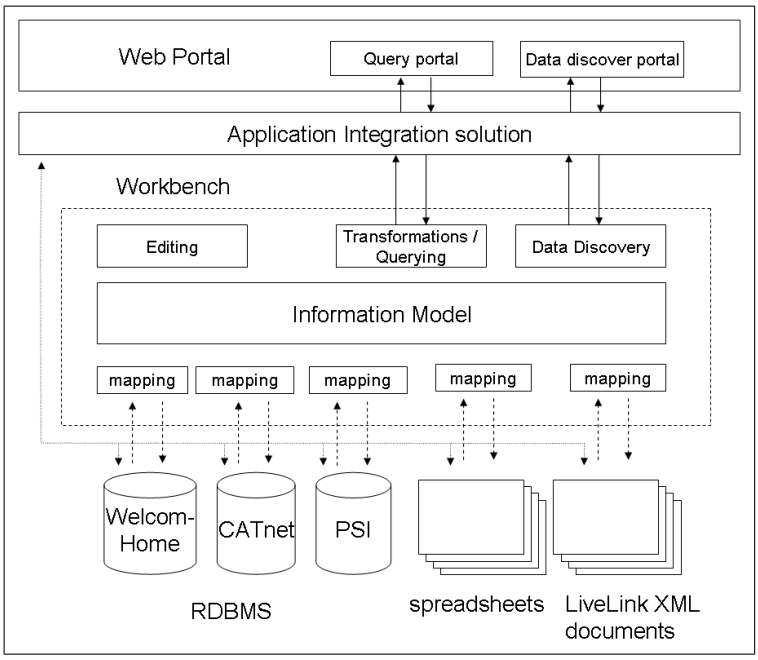

Figure 2.5: COG architecture

three relational databases, namely CATnet, PSI, and WelcomHome, one XML data sources, namely LiveLink, and a collection of Excel spreadsheet documents.

WelcomHome is an application used for project management (it supports the Microsoft Project application). LiveLink is a knowledge and document management system. CATnet and PSI are applications developed in-house at CRF to support the vehicle development and testing process.

CATnet has been developed to support the entire testing process, which is a major part of the FIAT vehicle (prototype) development process and essential for ensuring the quality of the products. Test requests that are linked to test plans are submitted to the CATnet system. These test plans support the planning of the execution of the tests, which is related to the vehicle development phase, the test laboratory, etc...

The test plans are linked to the so-called "technical memory", where the technical procedures for test execution are specified. This technical memory consists of PDF documents accessible through the LiveLink system. After the test executions, the test results are stored in the CATnet database for later retrieval. This retrieval of test results is critical in order to assure quality of the products. Whenever a customer complaint is received or a defect in a vehicle is detected, it has to be possible to easily retrieve the results of the tests performed on the concerning automobile component.

To facilitate this retrieval process, there is a customer perspective (also called the 'Voice of Customer' or VoC) tree for each vehicle model in the PSI system. The customer perspective trees are used to locate the appropriate tests that should have been performed on a particular vehicle system. The marketing manager can drill down the tree to locate the specific tests that have been performed, after which the test results can be retrieved from the CATnet system.

Besides these customer perspective trees that can be used for the retrieval of test results, the PSI system has functionality for creating packages of tests to be performed on a motor vehicle. These packages can be created by drilling down the customer perspective tree and selecting the desired standards (i.e. tests). These standards state what tests are to be executed on a certain vehicle system to ensure the quality of the product. Besides being maintained in the PSI system, some information about the standards and the customer perspective tree is being maintained in Excel spreadsheet documents.

In order to submit these testing packages to the CATnet system and in order to retrieve the appropriate test results from the CATnet database in the PSI system, these data sources need to be integrated. Besides the databases, the PDF documents (technical memory) and spreadsheet documents (concerning standards and the customer perspective tree) also need to be integrated to support the testing management.

Besides packets containing tests, there is also the concept of a session containing tests. While a packet can contain tests to be executed in different test labs, a test session is specific to one test lab and can contain test from different packages. These session are configured in the CATnet system, while the packets are configured in the PSI system.

It turns out that certain tests need to be executed in order to reach certain milestones in a project plan. Therefore, because information about project management is maintained in the WelcomHome database, this database also has to be integrated in order to integrate all aspects of the testing process.

2.4.2 Solving the integration problem in COG using the Semantic Information Management

During the development of the ontology in the COG project, some shortcomings in the Unicorn tool became apparent. In order to overcome these shortcomings, these problems were fed into the requirements for the new version of the Unicorn Workbench. During the COG project, a new version of Unicorn (v2.5) was released that overcomes most shortcomings in the old version identified during the development for COG. Examples of features developed especially for the COG project are the mapping to subtypes, specification of inverse properties, and mapping of foreign keys to inheritance relationships.

Another important development for the new version of Unicorn that has proven very useful for the COG project was the Asset API. Using this API it was possible to develop parsers for the integration of Excel documents and Microsoft Access databases (i.e. PSI) that can be used for importing data schemas from these platforms. Without this Asset API and these parsers, these sources could not have been integrated.

Based on the data sources provided by CRF and the interviews with domain experts at CRF, four main subject areas for the Information Model were identified, namely:

- *Motor vehicle development.* This area covers the motor vehicles themselves, such as automobile parts and vehicle systems.
- *Project management.* This area covers the project management capabilities present in WelcomHome.

- *Testing Management.* This area covers the management of the execution of the tests. Things like testing guidelines, test plan, testing laboratories, and so on, are the components of this area.
- *Test requests.* This area covers the customer perspective tree, standards, test requests, test results, etc ...

For each of these subject areas a package in the ontology was created, after which the classes, properties, and business rules were created corresponding to the composites in the source schemas. Interviews with domain experts were mostly used as input for the ontology development process, as well as working with the current applications (mainly PSI and CATnet) in use. This was the third phase (the construction phase) in the SIM methodology.

The next step (phase four - rationalization) was to map the source schemas to the ontology. This mapping was done using the mapping functionality of the Unicorn tool. During this mapping phase, many iterations back to phase three were necessary. It turned out that the classes in the ontology had been identified correctly, but still many properties and business rules had to be added or changed while trying to map the external assets.

During the mapping phase there were problems with the language of the source schemas. The most important sources were in Italian, but the designers for the ontology did not speak Italian and the ontology itself has been developed in English. This made it necessary to talk to the domain experts and especially the people in the organization knowledgeable about the data schemas.

The CATnet database was mainly mapped to the Test Management package, whereas the PSI database was mainly mapped to the Test Requests package. However, both databases are also to a great extent mapped to the other Test package (i.e. CATnet was also mapped to the Test Requests package and PSI was also mapped to the Test Management package). Besides this, PSI also contains some information about the Motor vehicle development.

The WelcomHome database was only mapped to the Project Management package, as it contains only information about the project management. The interdependencies with the other data sources are only expressed using the relationships in the ontology itself.

Below we provide some examples of how specific problems in the mapping of data schemas were solved in the COG project and how the Unicorn Workbench helped to structure and improve understanding of the data sources.

Mapping different types of data sources Different types of data sources (e.g. relational database, XML, Excel) have different data models and therefore not always be imported in a unified was. For example, a relational database has database tables along with database columns (called composites and atoms in Unicorn, respectively), where an XML document has elements with attributes, where elements can be nested in other elements. This nesting of elements can not be represented in the relational data model and furthermore, it is hard to express arbitrary relations between different elements (called 'foreign keys' in relational databases) in XML. These differences indicate the need for a different kind of internal representation (in the Unicorn Workbench) for different kinds of external assets.

Figure 2.6 shows the typical information for a table in a relational database. It shows the column names and data types as well as whether the column is

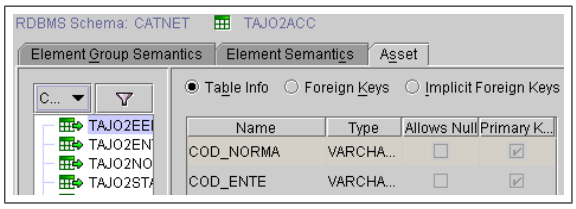

Figure 2.6: Example relational database external asset.

required and whether the column is part of the primary key. From the 'Table Info' view can be switched to the 'Foreign Keys' view in order to see the relationships between this table and other tables in the database, as well as the 'Implicit Foreign Keys' view, which shows the implicit foreign keys created within the Unicorn Workbench (see also Figure 2.9).

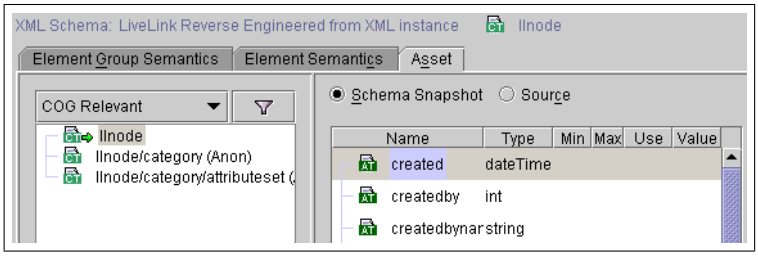

Figure 2.7: Example XML external asset.

Figure 2.7 shows the information displayed for a typical XML source. XML elements are listed on the left side, where `llnode` is the top element; the element `category` is nested within `llnode` and `attributeset` on its turn is nested in `category`. The right pane in the figure provides a list of attributes along with the (for XML Schema) typical properties, such as 'Type', 'Min', 'Man', etc. The other view presented here is the 'Source' view, in which the actual XML Schema can be viewed. However, the XML Schema is not very nice to look at (it's very complex and verbose), so we have decided not to show this view here.

Extracting inheritance relationships We can see an example of the application of the mapping to inheritance relationships feature in figure 2.8. Instances from the table `MSP_TEXT_FIELDS` in the WelcomHome database are mapped to the class `OfficialMileStone` in the ontology. There exists a foreign key in this table, called `FK_Official_Milestone_Tasks` (this is incidentally an implicit foreign key created in the Unicorn Workbench), that points to the `MSP_TASKS` table. This relationship maps to the inheritance relationship between the classes `Task` and `OfficialMilestone`, which is a sub-class of Task.

The aforementioned implicit foreign key that was created on the `MSP_TEXT_FIELDS` composite is illustrated in figure 2.9. The possibility of creating implicit foreign keys was very useful for the COG project, as many relationships in the source data schemas were not explicitly described using a foreign key.

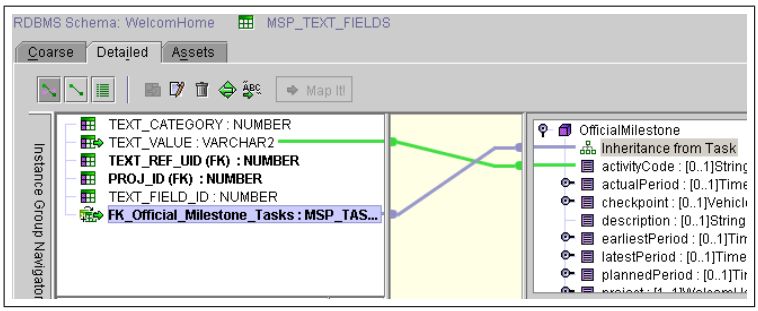

Figure 2.8: Mapping to the inheritance relationship; the generalization is made visible through mapping to the Information Model.

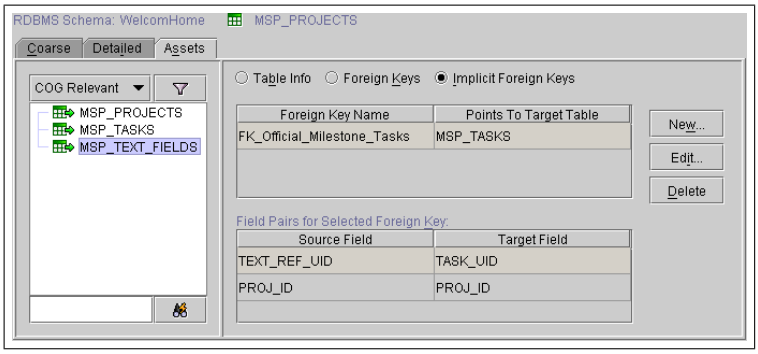

Figure 2.9: An Implicit Foreign key; created in the case of a missing foreign key in the asset schema.

Conditional mapping In figures 2.10 and 2.11 we see an example of the application of conditional mapping groups in the COG project. Figure 2.10 presents the coarse mapping view for the PSI relational database. Two filters have been applied to the view. First, the user has selected the 'COG relevant' subject group, which contains all composites relevant to the COG project ontology, so that only the relevant composites are displayed in the top-left pane. The second filter that has been applied filters out all mappings except the ones for the currently selected composite (this has been done by selecting the middle symbol above the middle pane at the top). A green line indicates the existence of a mapping from the composite to a certain class, the blue line indicates the currently selected mapping and the quadrangle symbol in a mapping indicates a conditional mapping. All the mappings in this particular view are conditional and an instance group has been created for each condition. In fact, each instance groups corresponds to one mapping. The instance groups are found in the bottom-left panel of figure 2.10 and the condition for the mapping is found in the bottom-right panel.

Mapping to indirect properties In figure 2.11 we see the (unfiltered) detailed mapping for one specific instance group (in this case 'Voice of customer CCP nodes'). We can see in this figure two examples of indirect mapping. The selected atom `tipo` has been mapped to the property name of class `Level` for

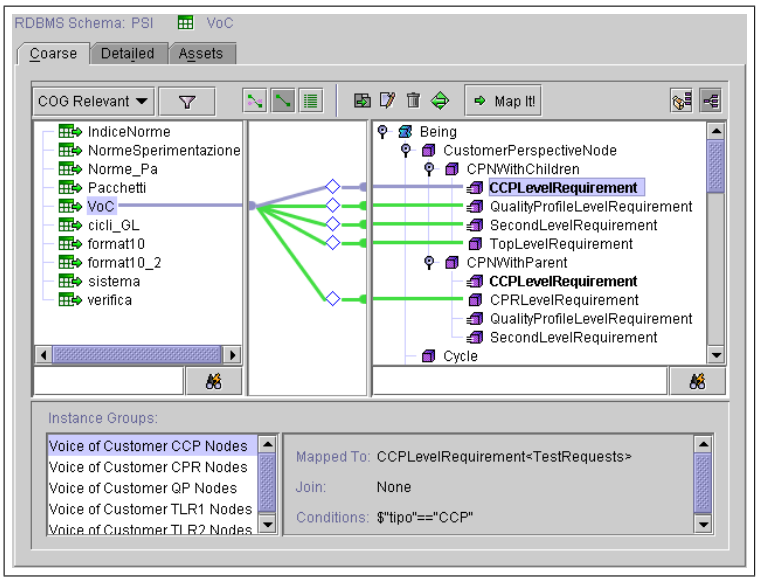

Figure 2.10: Conditional mapping in COG: on the basis of a condition, groups of instances in the flat database table are mapped to different classes in the ontology

property `level`. Furthermore, the atoms `p1` and `p2` have been mapped to an indirect property within `parent` (this is indicated by the dotted line in the right pane). If we now expand the `parent` property (figure 2.12) we can see how atoms `p1` and `p2` are mapped to the indirect properties `TLR1LevelAddress` and `TLR2LevelAddress`, respectively.

Inverse properties In the expanded view in figure 2.12 we also see a symbol with the text 'INV' to the left of the `child` property of `SecondLevelRequirement` and the child property of `TopLevelRequirement`. This symbol indicates that this property is an inverse of, in this case, the `parent` properties of `CCPLevelRequirement` and `SecondLevelRequirement`, repectively. This means that the parent of instance I_1 of `SecondLevelRequirement` must have I_1 in its set of children, and vice versa. The same goes for instances of class `SecondLevelRequirement`.

Mapping flat structures Figures 2.10 and 2.11 illustrate also how a very flat structure for a tree in a database can be mapped to a more explicit structure in an ontology. The instances in the VoC composite (in the PSI database) describe trees with a fixed number of levels. An identifier is used to indicate on which level the instance is located. For each level a class was created in the ontology. All these classes inherit from the class `CustomerPerspectiveNode`, as can be seen in the top-right pane in figure 2.10, and from either `CPNWithChildren` or `CPNWithParent`, or both. Using conditional mapping the appropriate class in the ontology is selected for the instance. In figures 2.11 and 2.12 we can see how a node in the tree is related to its parents via properties `p1` and `p2`. Remember that the `child` properties have been defined as inverse properties of

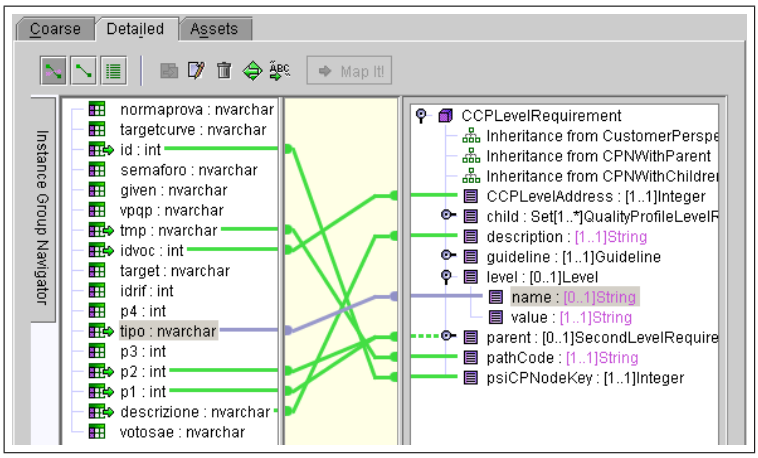

Figure 2.11: Detailed mapping

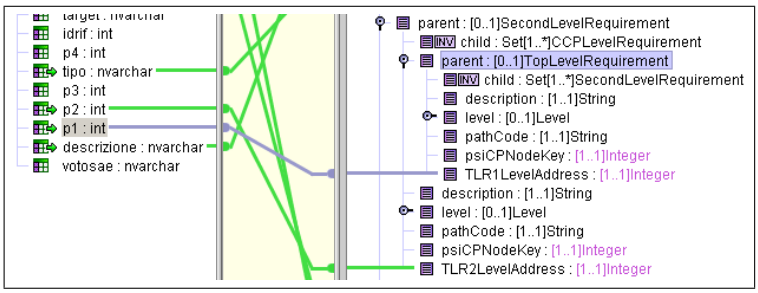

Figure 2.12: Mapping to indirect properties

parent properties, so that it is not necessary to explicitly map the relationship to the children.

We use these experiences gained from the COG project to compare the approach to other approaches in information integration in chapter 4.

2.5 Conclusions

In this chapter we have outlined the problems we faced in information integration in the COG project. We have described a solution to the integration problem using the Semantic Information Management Methodology [Schreiber, 2003] and the Unicorn Workbench tool, which we have applied in the COG project. We have compared this approach with several other approaches that are in different stages of development.

We have described how the SIM together with the Unicorn Workbench was used in the COG project and what the role is in the overall COG Architecture.

Many problems in the construction of the Information Model and the mapping to the disparate data schemas were caused by the poor understanding of the source data schemas. The data schemas contained concepts in the Italian language, while the ontology engineering and mapping was done by non-Italian speakers. What further complicated the matter was the fact that the users that

worked with the existing applications were no expert on the database schemas that were being used, which made the mapping a hard problem. It turned out that the only possibility for the ontology engineer to construct the ontology was to have a look at the applications together with the end-users, which was a tedious job.

These problems indicate the necessity of the usage of a central Information Architecture, through which the nature of the data residing throughout the organization can be understood.

Ontologies are very powerful in the sense that they are developed with the human understanding of the domain in mind, instead of taking a very application-oriented approach, as is mostly done with database schemas. Ontologies can help bridge the gap between human understanding and machine understanding of a domain [Fensel, 2003]. Using a tool such as the Unicorn Workbench, we can take the legacy database schemas in an organization, map them to a central Information Model, and in that way enable better understanding of this information by the users in the organization. Furthermore, because of the formal semantics of the Information Model, the Workbench is able to automatically derive instance transformations between external assets and to automatically translate queries on the Information Model to queries on individual data sources.

2.6 Limitations of the Unicorn Workbench and Future Work

We have identified several limitations of the Unicorn Workbench.

Lack of inter-operability One limitation of the Unicorn Workbench is the lack of inter-operability with other ontology engineering tools. It is not possible to import ontologies from existing often-used languages such as RDF(S), DAML+OIL and OWL. The export capabilities of the Workbench are also very limited. Only limited export to RDF(S) and DAML+OIL is possible.

Lack of ontology mapping possibilities Furthermore, it is currently not possible, in the Unicorn Workbench tool, to map different ontologies to each other[12]. Therefore, it is not possible to maintain several ontologies in an organization (only one central ontology fits in the architecture) and it is certainly not possible to cross organizational boundaries[13]. In order to enable the latter, inter-operability is required using standards-based languages, such as OWL, which is under development at the W3C standards body. It has to be possible to map the ontology created in the Workbench to other ontologies out there in order to allow inter-operation with other organizations (or even other organizational units).

[12]Although packages could be used to maintain different terminologies in an organization, as argued in section 2.1.

[13]It must be noted here that when an organization is powerful enough, it can impose a terminology on another organization. In this case ontology mapping is no longer necessary, because of the existence of a single standard.

One model for different types of sources A possible limitation is the fact that the same internal data model is used for the representation of different types of information sources in the *External Assets Layer*. This data model consists of concepts such as atoms, composites and foreign keys. This data model is clearly based on the relational data modelling practice in most current database management systems. This is not necessarily the best representation for XML-based, Excel-based and other kinds of data sources.

Lack of automation in creating the ontology and the mappings Another limitation is the very limited automation support in constructing the Information Model and creating the mappings to the external assets. Several algorithms and tools have been developed that aid in the construction of a central ontology and/or the detection of similarities of concepts in different schemas. Examples are PROMPT [Noy and Musen, 2000b], Anchor-PROMPT [Noy and Musen, 2000a], Chimæra [McGuinness et al., 2000], FCA-Merge [Stumme and Maedche, 2001], ARTEMIS [Castano et al., 2001; Bergamaschi et al., 2001] and SKAT [Mitra et al., 1999; Mitra et al., 2000]. We refer to [Rahm and Bernstein, 2001] for a comparison of matching algorithms.

We have to note here that we do not have any results of extensive tests of these algorithms and tools. Although most tools have been tested in limited environments (cf. [McGuinness et al., 2000; Noy and Musen, 2000b; Castano et al., 2001]), it is yet to be shown how these algorithms perform with large volumes of real-life database schemas.

In the next chapter we will look into a specific application of the Unicorn Workbench and the COG architecture. We will show how the Unicorn Workbench can be used to create reusable conceptual queries on an ontology and how this query is automatically translated to the native query language of the target database platform.

Chapter 3

Database Querying using Ontology Technology

This chapter[1] is partly based on a White Paper to be published as a part of the COG project at http://www.cogproject.org/, with the title 'Active Ontologies for Data Source Queries' [de Bruijn and Lausen, 2003].

The aim in the COG project, as described in the previous chapter, was to create a Semantic Information Management [Schreiber, 2003] in which several heterogeneous data sources are integrated, using Semantic Web technology, into one global unified view. This global view enables the user to discover data residing at different location in the organization, to automatically transform instance data from one representation to another, and to query instance data residing in the disparate data sources. This chapter focuses on the latter. We discuss the querying task, the querying support in the COG Architecture, and the translation of conceptual queries on the Information Model to queries on individual database (and other) schemas.

In this chapter, we will first introduce the querying support in the Unicorn Workbench tool (part of the Unicorn System), thereby enhancing the description given in the previous chapter, in section 3.1. Then, we will show how this querying support was leveraged in the architecture of the COG project and we will show the extensions of this querying capability that were required for the COG project, in section 3.2. Finally, we provide some conclusions in section 3.3.

3.1 Querying disparate data sources using the Unicorn Workbench

In this section, we present the support in the Unicorn Workbench tool for the conceptual querying of an ontology (the Information Model) and the subsequent translation to asset queries. With asset queries we mean queries that can actu-

[1] Some materials presented in this chapter are the copyright of Unicorn Solutions, Inc. and are used with permission.

ally be executed on the native platforms of the external assets that have been mapped to the central ontology in the Unicorn Workbench.

The Unicorn Workbench has a facility to create SQL-like queries on the ontology. In the query, a number of properties are selected from a certain class using a boolean condition (specified in the 'where' clause). The condition can be created using the Unicorn conversion language [Unicorn, 2003a], which is also used for the specification of business rules[2] in the Unicorn Workbench (see section 2.3). The Unicorn conversion language is based on the Python language and enables transformation of data values and basic boolean operations (e.g. equality, greater-then, etc.).

These queries are issued against a single relational database and are translated automatically by Unicorn into a correct SQL statement. This SQL statement can be automatically retrieved (via the operational API) by the (custom) run-time architecture or can be manually executed in the source database system. The COG architecture, presented in section 3.2.1, contains such a run-time engine, which retrieves parameterized queries from the Unicorn Workbench.

We distinguish two main Use Cases for creating queries in the Workbench:

- During *design-time* the ontology can be manually evaluated by creating queries on the ontology, translating them to SQL queries for the source database platforms and execute them to verify that the expected results are returned.
- During *run-time*, when an information architecture is set up, using the Information Model in the Unicorn Workbench, queries can be created on the Information Model, where the Unicorn Workbench would translate them into the native SQL format used in the databases. The middle-ware would then take care of executing the query in the database platform and retrieving the results for the front-end application.

In this section we first introduce the querying capabilities presented by the Unicorn Workbench, after which we show the translation of conceptual queries into actual SQL queries, which can be executed on the native database platform. We finish with discussing some limitations of the querying functionality in the Workbench.

3.1.1 Queries in the Unicorn Workbench

Unicorn has developed a conceptual query language for querying ontologies in the Unicorn Workbench. With this query language, the user can specify conceptual queries on classes in the ontology (Information Model). These conceptual queries are, with the use of mappings to database schemas, automatically translated into instance queries that can be executed on the database platform of the data source. Queries created in the Unicorn Workbench are instance queries; only queries on instances can be created. It is as yet not possible to query the ontology itself.

The query itself consists of the following five parts:

- A *name*, identifying the query.

[2]Business rules restrict the values of properties by specifying a relation between properties or by restricting the value of the property to a limited set of possible value.

- A *select* clause, which specifies which (indirect) properties to retrieve. These properties can only be of fundamental data types and can be converted using the Python-based Unicorn Conversion Language [Unicorn, 2003b].
- A *from* clause, which specifies which class in the ontology is queried.
- A *where* clause, which specifies which additional conditions the instances need to satisfy to be selected. Here also the Unicorn Conversion Language can be used for transforming values.
- An *on database* clause, which specifies which database to query.

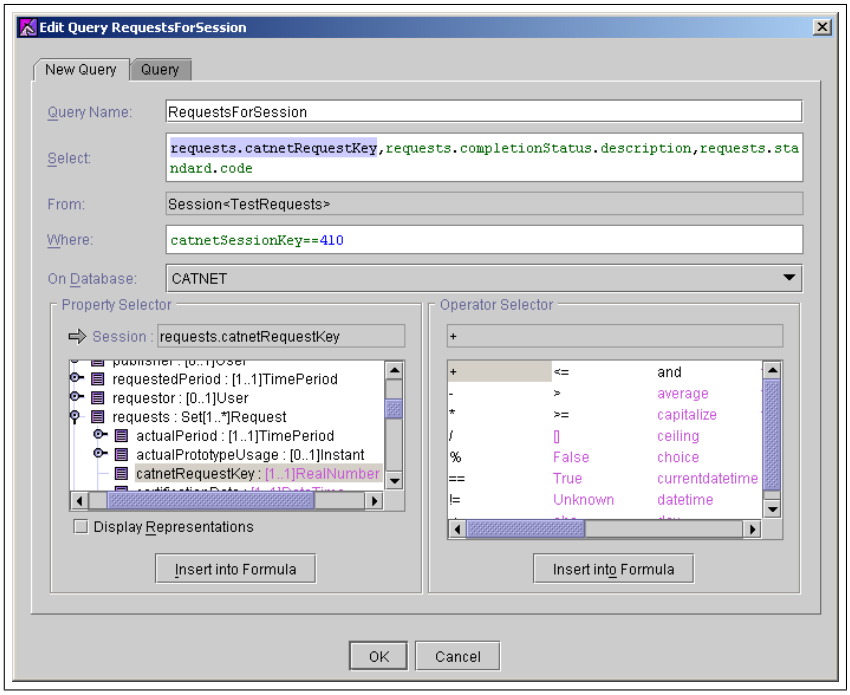

Figure 3.1: Example query in the Unicorn Workbench

An example query is shown in Figure 3.1. At the top of the figure we see the query name, which is `RequestsForSession`. Below that we see the properties to be selected, under the heading 'Select'. One of the selected properties is highlighted, namely the property `requests.catnetRequestKey`. Note that this same property is selected in the property selector at the bottom left of the figure. We must also note here that only properties that refer to classes in the "fundamental" package[3] (see [de Bruijn et al., 2003a]) can be selected here. This is because the query is translated to a SQL query, where only built-in data types can be selected. Note that because of the mechanism of indirect properties, it is possible to select properties that belong to properties of the class on which the query is performed, as indeed is done here with the `catnetRequestKey` property

[3]The fundamental package is shipped with the Unicorn Workbench and contains fundamental data types such as integer, string, date, etc...

of the `Request` class, which is the type of the `requests` property in the `Session` class, on which we perform the query.

The next caption, 'From', is fixed to `Session` (the 'Session' concept will be explained in more detail in section 3.2) in the example, because in the Workbench a query is performed on one class. In our example, the 'Where' clause specifies the condition that the value of the `catnetSessionKey` property is '420'. The last part, 'On Database', contains the database on which we want to perform the query. A query can only be translated to the native SQL format of one database, so the database on which the query is to be executed, must be selected. In our case we chose to query the database `CATNET`.

The panel at the bottom left of the query editing window is the Property Selector, which can be used to conveniently select properties from the class (and related classes) to be used in the query, thereby saving lookup time and preventing typing mistakes. The panel at the bottom right is the Operator Selector, where boolean, string, and integer operators can be selected, which can be used in the 'Select' and the 'Where' clauses. In fact, in Figure 3.1 we already see the boolean equality operator put to use in the 'Where' clause to check the equality of the `catnetSessionKey` property and the number `420`.

In order to enable reuse of queries in different settings, the queries allow for parameterization. An arbitrary number of parameters can be used in a query; these parameters are replaced with actual values when the queries are extracted during run-time. For example, we can replace the value '420' in Figure 3.1 with a generic parameter, allowing us to reuse the query for different sessions (a session is uniquely identified by the `catnetSessionKey`). In section 3.2.1 we show how this parameterization works in practice.

3.1.2 Transforming conceptual queries into database queries

There is no query execution engine in Unicorn; instead, the query is translated into a SQL query, which can be manually executed in an external database system. The fact that the query can currently only be translated to a SQL database query is a limitation. The wrappers currently used for the Asset API do not support the translation of queries to the native platform. This translation is currently done by a separate component, which limits generality of the query.

There are, however, efforts under way to generate Query Planners, in the same way Transformation Planners[4] work for transformations.

Queries that are created are stored in the Unicorn project along with the SQL translation and possibly some descriptor information provided by the user. The queries can be updated when the ontology or the mappings to the data assets are updated. Unicorn will warn the user when a query has possibly become invalid.

In figure 3.2 we see the SQL translation of the ontology query shown in figure 3.1. We see here that the SQL query has been annotated with comments that specify the parts in the query that correspond to the properties that were specified in the ontology query.

[4] A Transformation Planner is an XML document describing which composites and atoms from the source schema are to be mapped to which composites and atoms from the target schema. This XML document can be used to develop the actual transformation.

```
SELECT
    CATNETREQUESTKEY AS BR    /* BR is business rule
                                 requests.catnetRequestKey */,
    DESCRIPTION AS BR1    /* BR1 is business rule
                             requests.completionStatus.description */,
    CODE AS BR2    /* BR2 is business rule requests.standard.code */
FROM
    (
    SELECT
        A.AK_7RCH AS CATNETREQUESTKEY    /* CATNETREQUESTKEY is property
                                            requests.catnetRequestKey */,
        B.DESCRIZIONE AS DESCRIPTION    /* DESCRIPTION is property
                                           requests.completionStatus.description */,
        A.COD_NORMA AS CODE    /* CODE is property requests.standard.code */,
        C.AK_7SEL AS CATNETSESSIONKEY    /* CATNETSESSIONKEY is property
                                            catnetSessionKey */
    FROM
        COG.TAJO7RCH A,
        COG.TAJO2STA B,
        COG.TAJO7SEL C
    WHERE
        C.AK_7RCH = A.AK_7RCH AND
        A.COD_STATO_RICHIESTA = B.COD_STATO_RICHIESTA AND
        B.COD_STATO_RICHIESTA = A.COD_STATO_RICHIESTA
    ) SESSION    /* SESSION is class Session in package TestRequests */
WHERE
    CATNETSESSIONKEY=410
```

Figure 3.2: Example SQL translation of an ontology query

Let's try and reconstruct the SQL as is presented in Figure 3.2. We see a nested `SELECT` statement, where the inner statement is used to reconstruct, using a join operation[5], the ontology class `Session`, including the required indirect properties, from the tables in the database that have been mapped to the class.

In the outer `SELECT` statement the requested properties are selected from the inner statement using the 'Where' condition specified in the ontology query and translated to the database platform (Oracle in this case). Note that only the atoms necessary in the outer `SELECT` statement are selected.

3.1.3 Limitations of the current approach

A disadvantage of the query support in the Unicorn Workbench is that only fundamental data types (i.e. integer, string, etc...) can eventually be queried. This means that the user usually has to drill down the indirect properties in order to find the required data type to be retrieved from the database. When that specific data value happens to be residing in a different database, it is not possible to retrieve it in the same query. The user needs to create a new query in order to retrieve the value from the other database.

With the current version (2.6.1) of the Unicorn Workbench it is not possible to use a single query for multiple databases. In fact, only relational databases can be queried with the Unicorn tool at the moment. To query multiple databases, it is necessary to create the same conceptual query several times, where each query differs only in the data source to which it refers. This creates maintenance problems.

[5]Notice that the last two clauses in the 'WHERE' clause of the inner SELECT statement are actually equivalent and thus redundant. This is apparently a bug in the current version of the software; this bug will not cause many problems because any optimizer will filter out such redundancies before query execution.

Another drawback of the current version of the Unicorn tool is that it is not possible to automatically retrieve the results of a query from a data source. In this scenario it also doesn't make any sense to query multiple databases at the same time. This only makes sense if multiple sources are automatically queried and if the results are integrated into a unified view for the user. In the querying scenario as envisioned by Unicorn, the run-time architecture will take care of the querying of the databases and the integration of the results into a unified view for the user. In this case, the run-time architecture can use the Unicorn Query API to specialize the queries for the different databases from which the query results need to be integrated.

We have presented above the query support in the Unicorn Workbench tool. We have presented the conceptual querying language, used to issue queries on a class of the ontology. Then we have looked into the way the conceptual query was translated into an actual SQL query that can be executed on a native database platform. Finally, we have looked into some limitations in the querying support of the Unicorn Workbench. In the next section we will look into the way the query support in the Workbench was used in the COG project and how the COG architecture overcomes some of the limitations of the Workbench.

3.2 Querying disparate data source in the COG project

One of the goals of the COG project was to create a platform-independent semantic querying tool to query the disparate data sources through the central ontology. The ontology has been implemented using the Unicorn Workbench, a tool supporting the Semantic Information Management ([Schreiber, 2003]). Unicorn provides basic ontology querying support, which was described in detail in the previous section. The querying support is further expanded in the COG run-time architecture.

3.2.1 The querying architecture in the COG project

In the COG project, a web client was created as a front-end application for the Unicorn semantic layer. The web client integrates the various disparate sources using the semantic back-bone provided by the Unicorn run-time API. Figure 3.3 shows the disparate data sources in the COG project that are integrated using the central ontology in the Unicorn Workbench.

One of the goals in the COG project is to provide support for semantic platform-independent querying of the disparate data sources through the central ontological model.

In the overall COG architecture (figure 3.4), within the application integration layer, there is the Query Manager, which interacts with the Unicorn Workbench and with the individual data sources in order to automatically execute queries, created using the Information Model, on the disparate databases. There are two active components in the query execution process. The Query Manager communicates with the Unicorn tool to retrieve the Query object for a specific query. This Query object translates the conceptual query from the Uni-

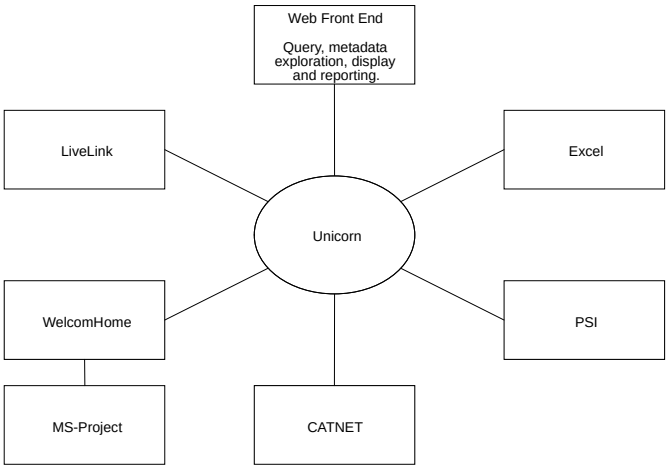

Figure 3.3: COG logical architecture

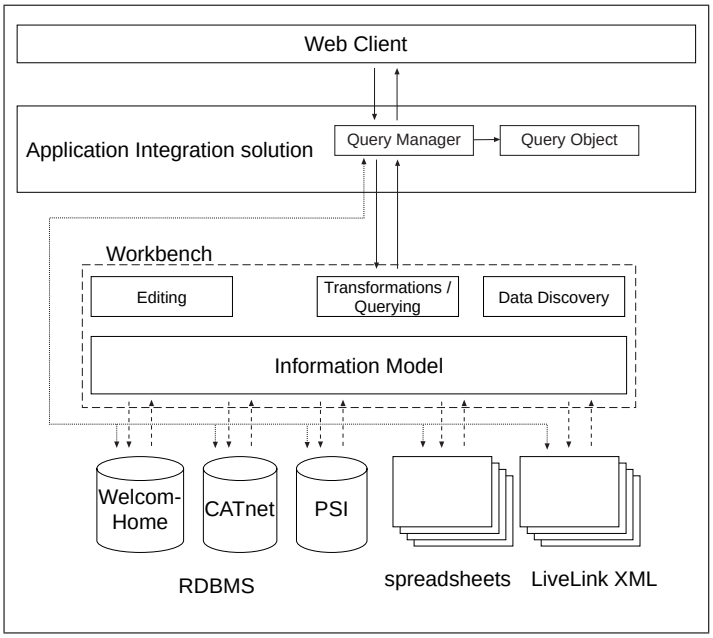

Figure 3.4: COG Querying architecture

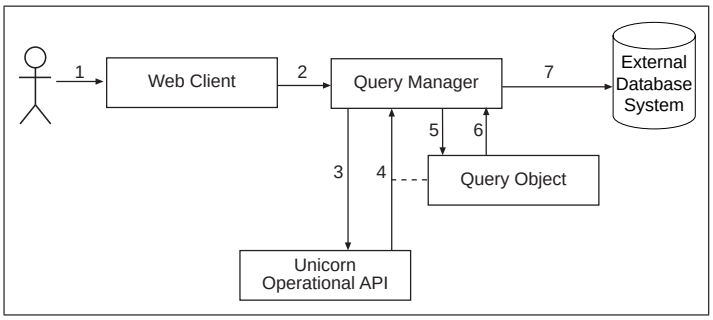

Figure 3.5: The querying process in the COG architecture

corn Workbench into the query language that is understood by the individual data source.

The querying process is depicted in figure 3.5. The web client sends a request to the Query Manager, which in turn retrieves the corresponding SQL query from the Unicorn Operational API, after which the Database System is contacted and the query is executed. The query results are then propagated back to the web client.

The following steps are involved in the querying process:

1. The user selects the query to be executed along with its parameter values in the Web Client.
2. The Web Client identifies the query and sends the query name along with the parameter value to the Query Manager.
3. The Query Manager will use the Unicorn API to request the query.
4. The Unicorn API returns a Query Object, which represents the conceptual query.
5. The Query Manager sends the parameter values to the Query Object.
6. The Query Object returns the platform-native translation (typically SQL) of the conceptual query after having performed the parameter-value substitutions.
7. The query is sent to the database system, where it is execute. The results are returned, translated to a suitable format by the Query Manager, and ultimately displayed to the user by the Web Client.

We have shown in this section how the querying of databases works and what the role is of the Unicorn Workbench in the querying process. In the next section we clarify the querying architecture using some examples from the user's point of view.

3.2.2 Querying in the COG showcase

In figure 3.7 we see a screen-shot[6] from the COG showcase where the user can select an item at a certain level in the Voice of Customer (VoC) tree for a specific

[6]Note that the screen-shots shown here are taken from an HTML prototype of the showcase, since the real showcase was not finished at the time of writing.

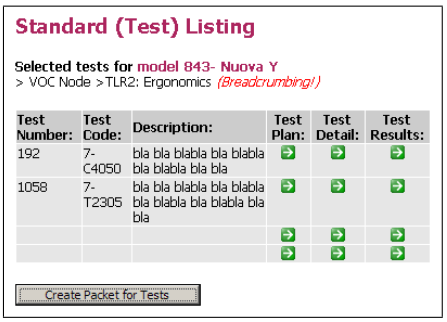

Figure 3.6: VoC node listing example

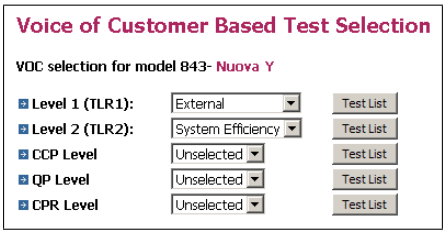

Figure 3.7: VoC selection example

prototype model (in this case 'model 843- Nuova y'). The user selects an item from one of the drop down lists and presses the 'Test List' button. The web portal now contacts the application server to execute a query with a specific name with (a) specific parameter(s). The Query Manager component (figure 3.4) now contacts the Unicorn Workbench API in order to retrieve the query. After the query has been retrieved, it is executed on the target database.

In figure 3.6 we see a list of tests corresponding to a particular node in the VoC tree (in this case a `Level 2` node called 'Ergonomics'). The ontology query used to retrieve the list is shown in figure 3.8. The list contains of some information about the test (the test number, code and description) and also links to more detailed information. This detailed information, consisting of the test plan, test details, and test results, is retrieved using consecutive querying. For each of these actions a parameterized query has been created in the Unicorn Workbench. The application server retrieves the query from the Workbench, fills in the parameter, and issues the query on the database system.

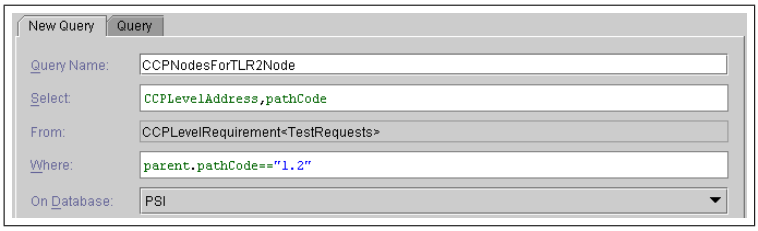

Figure 3.8: Level 2 node selection query

The retrieval of test details using the Voice of Customer tree is usually done for the retrieval of test results, when defects in the prototype have surfaced or when customer complaints reach the marketing manager.

During the creation of test plans and the execution of tests, two views are very important. First, there is the packet view (figure 3.9), which is used to create packages of tests to be performed on a model. This view is used by the prototype manager to select a set of tests based on customer requirements. The second view is the session view (figure 3.10), which is used by the testing manager, who configures testing sessions to be performed by specific testing laboratories. To summarize, a package is a set of tests from the customer requirement

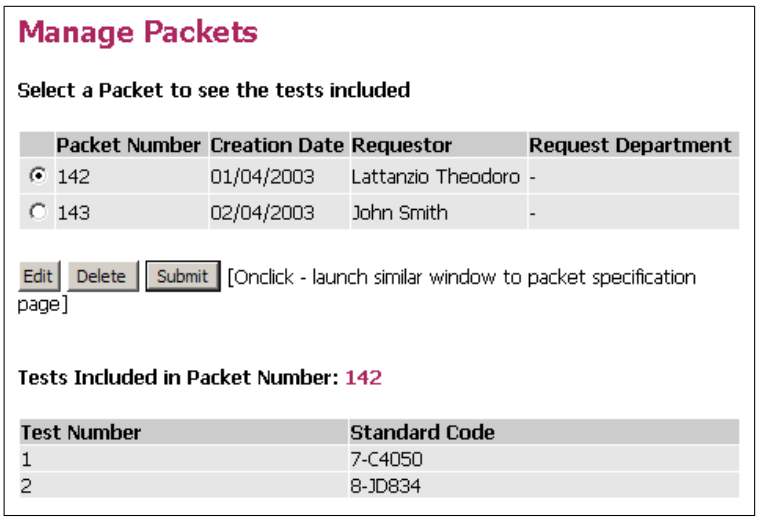

Figure 3.9: Example packets in the package management section of the COG showcase

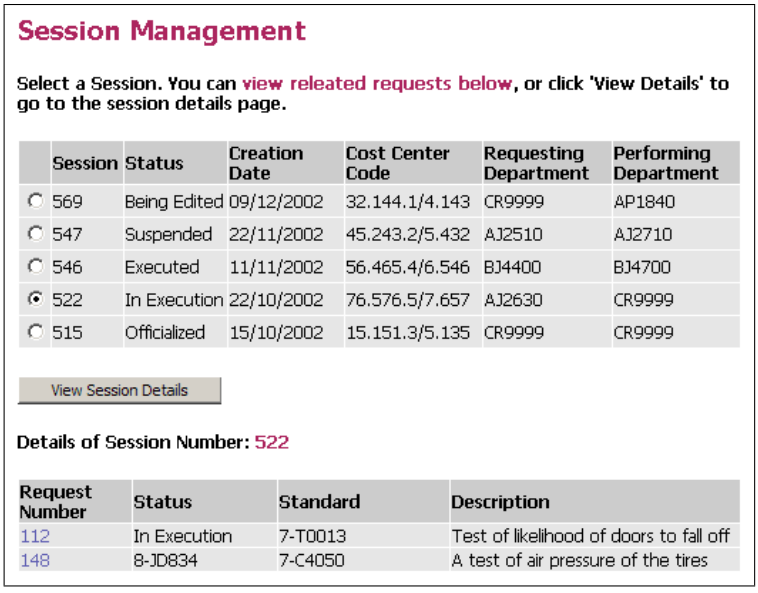

Figure 3.10: Example sessions in the session management section of the COG showcase

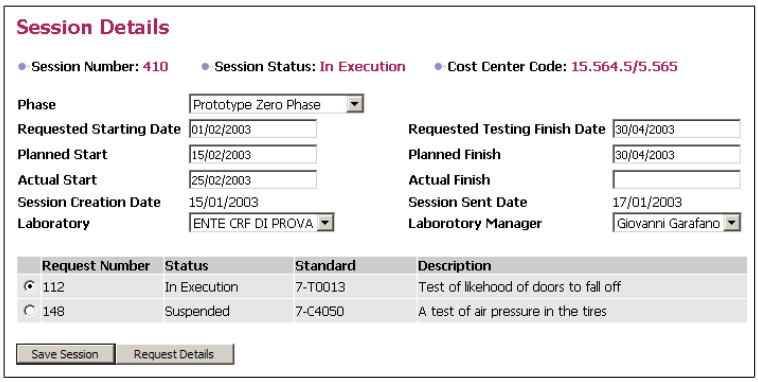

Figure 3.11: Testing Session details

point-of-view and a session is a set of tests (figure 3.11) to be performed by a specific laboratory, not specifically related to customer requirements.

We have shown above how conceptual queries, created in the Unicorn Workbench, are used in an actual application. We presented the COG architecture, which support retrieving SQL queries, based on conceptual queries, from the Workbench, executing them on the native database platforms, and retrieving the query results for the front-end application.

3.3 Conclusions

In this chapter, we have evaluated the ontology querying capabilities in the Unicorn Workbench, as well as the possibilities of generating SQL queries on physical database schemas that have been mapped to the Information Model.

We have furthermore analyzed how this querying functionality is used in the COG architecture. We have seen how conceptual queries originating from the web front-end application are translated by the Unicorn Workbench into queries that are ready to be executed on the native database platforms; how these queries are executed by the Query Component and how the results are returned to the front-end, which displays them to the user.

Using the Unicorn Workbench to retrieve the queries prevents problems that usually arise when data schemas are updated. The applications using these data schemas will usually break and be rendered useless. By storing the queries in the Workbench, queries are maintained with the ontology and the mappings to the database schemas. When the Information Model and the mappings to the external assets are maintained within the Workbench, the maintainer will be warned when invalidating existing queries and will be inclined to (automatically) update these queries to reflect the changes in the Information Model and the external assets.

In the next chapter we conduct a survey of existing methods and tools in the area of semantic information integration and ontology mapping and compare these with the information integration solution provided in the COG project.

Chapter 4

Comparison with other initiatives

In this chapter we evaluate several approaches to semantic information integration and compare them with the approach we used in the COG project.

The approaches used for the comparison consist mainly of research projects in the semantic (mostly ontology) integration area. Many of these approaches specifically integrate ontologies and provide translations of database schemas to ontologies in order to integrate them. Examples of this approach are MOMIS [Bergamaschi et al., 2001] and ONION [Mitra et al., 2000].

Two tools included in the comparison, PROMPT [Noy and Musen, 2000b] and Chimæra [McGuinness et al., 2000], are specialized in ontology merging and do not provide an architecture for run-time translation or transformation of database schemas to ontologies. However, these tools have been developed some time ago and have evolved to incorporate a lot of research done in the ontology integration area in the last few years.

This chapter is organized as follows: section 4.1 provides a survey of selected methods and tools for Semantic Information Integration and section 4.2 compares the approaches in the survey and the approach in the COG project, described in chapter 2.

4.1 Methods and Tools for Semantic Information Integration

In this section we will describe several approaches to semantic information integration using ontologies, as well as the tools available for these approaches.

We will classify each approach using the classification scheme outlined in the introduction. We make the distinction between merging and aligning, the distinction between local model and global model and the distinction between one-to-one mapping, using a central ontology and ontology clustering.

Furthermore, we will make a distinction in the degree of automation supported by the tool. We distinguish manual, semi-automatic, and automatic schema integration. With manual integration, the user will have to discover all merging candidates in the source schemas that need to be mapped or merged in

to the global schema. When semi-automatic integration is supported, the user is typically provided with suggestions for integration. There are currently no algorithms or tools that support automatic integration.

We do not expect to see automatic integration tools appear, because of the grave differences, both on a syntactic and a semantic level, between different data schemas (and ontologies) that have been developed and will be developed. Schemas and ontologies are created by different people with different views on the domain. These different views cannot be automatically integrated and will very likely require negotiations between stakeholders who have these different views.

We restrict this survey to methods and tools for schema integration. We will not evaluate general methodologies for ontology engineering, because these are outside the scope of this thesis. We refer the interested reader to [Fernández-López, 1999] and [Fernández-López et al., 2002]. For a survey on ontology development tools, we refer the reader to [Gómez-Pérez et al., 2002].

In this section we will give a short description of each of the methods, describe the tool support and give a short summary. We have included the tool support in order to show the reader how the user is supported in the integration task in the presented methodology.

A selection of these methodologies has been used for a comparison with the methodology employed in the COG project.

4.1.1 The MOMIS approach

An approach to the integration of heterogeneous data sources is the MOMIS (Mediator envirOnment for Multiple Information Sources) approach [Bergamaschi et al., 1999; Bergamaschi et al., 2001].

The goal of MOMIS is to give the user a global virtual view of the information coming from heterogeneous information sources. MOMIS creates a global mediation schema (ontology) for the structured and semi-structured heterogeneous data sources, in order to provide to the user a uniform query interface.

The first step to the creation of the global mediation schema is the creation of the Common Thesaurus from the disparate data sources. To do this, first a wrapper is created for each data source in the ODL_{I^3} [Bergamaschi et al., 2001] language. ODL_{I^3} is an object-oriented language with an underlying Description Logic [Baader et al., 2003] language OLCD [Bergamaschi et al., 2001], which enables making inferences (e.g. subsumption) about the classes expressed in that language.

Using the disparate schemas, a Common Thesaurus is created, which describes intra- and inter-schema knowledge about ODL_{I^3} classes and attributes of source schemas. The Common Thesaurus is built in an incremental process in which relationships (between classes) are added based on the structure of the source schemas, lexical properties of the source classes and attributes (e.g. WordNet [Fellbaum, 1999] can be used to identify synonyms), relationships supplied by the designer, and relationships inferred by the inference engine.

Once the Common Thesaurus has been created, a tree of affinity clusters is created, in which concepts are clustered based on their (name and structural) affinity. The name affinity coefficient is calculated based on the terminological relationships between two classes. The structural affinity coefficient between two classes is calculated based on the level of matching of attribute relationships in

the Common Thesaurus. The sum of these two coefficients is the global affinity coefficient, which is used to construct the affinity tree, in which concepts with a high affinity are clustered together.

For each cluster in the affinity tree, a global class is (interactively) created. For each global class a mapping is maintained to all the source classes.

A number of components are used to enable the MOMIS architecture. These components are (see Figure 4.1, taken from [Bergamaschi et al., 2001]):

- A *wrapper* performs the translation of the individual data source into the ODL_{I^3} language (and translates the queries back).
- The *mediator* consists of the query manager (QM) and the global schema builder (GSB). The QM component breaks up global ODL_{I^3} queries into sub-queries for the different data sources.
- The *ARTEMIS* tool environment performs the classification (affinity and synthesis) of classes for the synthesis of the global classes.
- The *ODB-tools engine* performs schema validation and inferences for the generation of the Common Thesaurus.

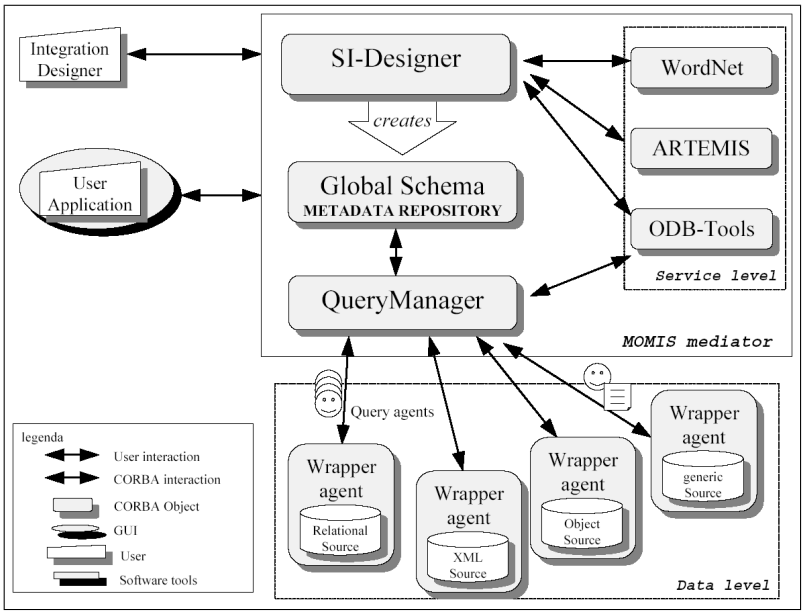

Figure 4.1: Architecture of the MOMIS system

Tool support The architecture in figure 4.1 (taken from [Bergamaschi et al., 2001]) shows the main tools used to support the overall architecture. A disadvantage is that there is no integrated tool environment.

Summary The MOMIS approach is a semi-automatic approach to schema integration, developed at the university of Modena, Italy. The approach has not been used in any major industrial project and is mainly an academic research

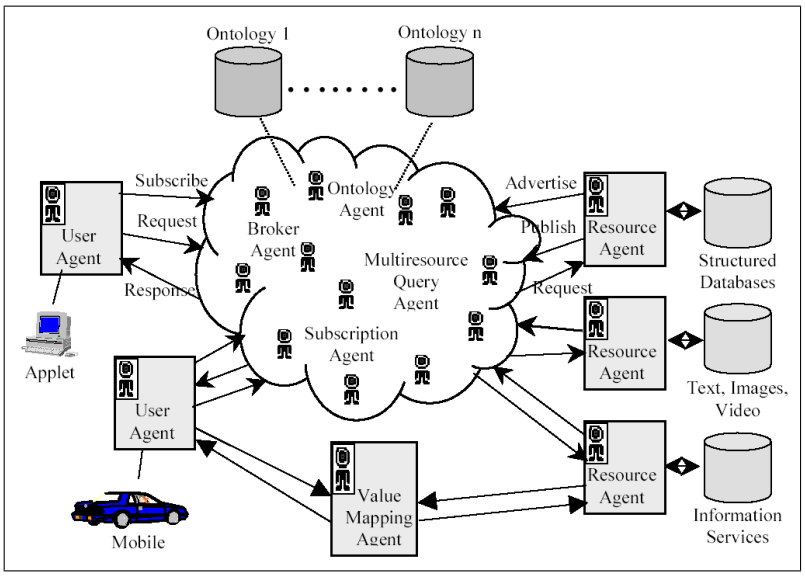

Figure 4.2: The InfoSleuth architecture

activity. Any data source can be connected to the architecture, as long as an ODL$_{I^3}$ wrapper is created. MOMIS has a single mediator, which provides a global data schema and query interface to the user.

4.1.2 InfoSleuth

InfoSleuth [Fowler et al., 1999] is an agent-based system, which supports construction of complex ontologies from smaller component ontologies so that tools tailored for one component ontology can be used in many application domains. Examples of reused ontologies include units of measure, chemistry knowledge, geographic metadata, and so on. Mapping is explicitly specified among these ontologies as relationships between terms in one ontology and related terms in other ontologies.

All mappings between ontologies are maintained by a special class of agents known as resource agents. A resource agent encapsulates a set of information about the ontology mapping rules, and presents that information to the agent-based system in terms of one or more ontologies (called *reference ontologies*). All mapping is encapsulated within the resource agents. Ontologies are represented in OKBC (Open Knowledge Base Connectivity) [Chaudhri et al., 1998] format and stored in an OKBC server by a special class of agents called ontology agents, which provide ontology specifications to users (for request formulation) and to resource agents (for mapping).

The InfoSleuth architecture [Nodine et al., 2000] (figure 4.2) consists of a number of different types of agents:

- The *user agents* act on behalf of the user and maintain the users state. They provide a system interface that enables users to communicate with the system.

- The *resource agents* wrap and activate databases and other repositories of information. They translate queries and data stored in external repositories between their local forms and their InfoSleuth forms. There are resource agents for different types of data sources, including relational databases, flat files, and images.
- *Service agents* provide internal information to the operation of the agent system. Service agents include *Broker agents*, which collectively maintain the information the agents advertise about themselves, *Ontology agents*, which maintain a knowledge base of the different ontologies used for specifying requests, and *Monitor agents*, which monitor the operation of the system.
- *Query and analysis* agents fuse and/or analyze information from one or more resources into single (one-time) results. Query and analysis agents include *Multi-resource query agents*, that process queries that span multiple data sources, *Deviation detection agents*, that monitor streams of data to detect deviations, and other data mining agents.
- *Planning and temporal agents* guide the request through some processing which may take place over a period of time, such as a long-term plan, a workflow, or the detection of complex events. Planning and temporal agents include *Subscription agents*, that monitor how a set of information (in a data source) changes over time, *Task planning and execution agents* plan the processing of user requests in the system, and *Sentinel agents* monitor the information and event stream for complex events.
- *Value mapping agents* provide value mapping among equivalent representations of the same information.

Tool support There are Java templates available to make the development of new agents easier. To create a resource agent using such a template, it is in general sufficient to just supply a configuration and a mapping file to complete the agent [Nodine et al., 2000]. It is possible to use different ontologies in an InfoSleuth system. Each OKBC-compliant [Chaudhri et al., 1998] Knowledge Base can be used in InfoSleuth by wrapping it using an *ontology agent* [Nodine et al., 2000].

Summary InfoSleuth is a multi-agent system for semantic inter-operability in heterogeneous data sources. Agents are used for query and instance transformations between data schemas. An agent is aware of its own ontology and the mapping between that ontology and the data schema, it is aware of the shared ontologies and it can map its ontology to those of other agents. InfoSleuth uses several shared ontologies, made available through the ontology agents. Individual data sources have (through the resource agents) a mapping to these shared ontologies. The shared ontologies are linked together through one-to-one ontology mapping. Note that the user agents use the shared ontologies as their vocabulary and local ontologies are only maintained by the resource agents.

4.1.3 OBSERVER

OBSERVER (Ontology Based System Enhanced with Relationships for Vocabulary hEtereogenity Resolution) [Mena et al., 2000] combines intensional and extensional analysis to calculate lower and upper bounds for the precision and recall of queries that are translated across ontologies on the basis of manually identified subsumption relations. The system uses a component-based approach to ontology mapping. It provides brokering capabilities across domain ontologies to enhance distributed ontology querying, thus avoiding the need to have a global schema or collection of concepts.

OBSERVER uses multiple pre-existing ontologies to access heterogeneous, distributed and independently developed data repositories. Each repository is described by means of one or more ontology expressed in Description Logics (DL). The information requested to OBSERVER is expressed according to the user's domain ontology, also expressed using DL. DL allows matching the query with the available relevant data repositories, as well as translating it to the languages used in the local repositories.

The system contains a number of component nodes, one of which is the user node. Each node has an ontology server that provides definitions for the terms in the ontology and retrieves data underlying the ontology in the component node. If the user wants to expand its query over different ontology servers, the original query needs to be translated from the vocabulary of the user's ontology into the vocabulary of another's component ontology. Such translation can not always be exact, since not all the abstractions represented in the user ontology may appear in the component ontology. If this is the case the user can define a limit in the amount of *Loss of Information*. Anyhow, the user can always set this parameter to 0, thereby specifying no loss at all.

An Inter-ontology Relationship Manager (IRM) provides the translations between the terms among the different component ontologies. The IRM effectively contains a one-to-one mapping between any two component nodes. This module is able to deal with *Synonym*, *Hyponym*, *Hypernymy*, *Overlap*, *Disjoint* and *Covering* inter-ontology relationships.

The user submits a query to the query processor in its own component node (in fact, each component node has a query processor). The query processor uses the IRM to translate the query into terms used by the other component ontologies and retrieves the results from the ontology servers.

Tool support The OBSERVER architecture, depicted in figure 4.3 (taken from [Mena et al., 2000]), consists of a number of component nodes and the IRM node. A component node contains an *Ontology Server* that provides for the interaction with the ontologies and the data sources. It uses a repository of mappings to relate the ontologies and the data sources and to be able to translate queries on the ontology to queries on the underlying data sources. The architecture contains one Inter-Ontology Relationship Manager (IRM), which enables semantic inter-operation between component nodes by maintaining the relationships between the ontologies.

Summary OBSERVER is an architecture consisting of component nodes, each of which has its own ontology, and the Inter-ontology Relationship Manager (IRM), which maintains mappings between the ontologies at the different

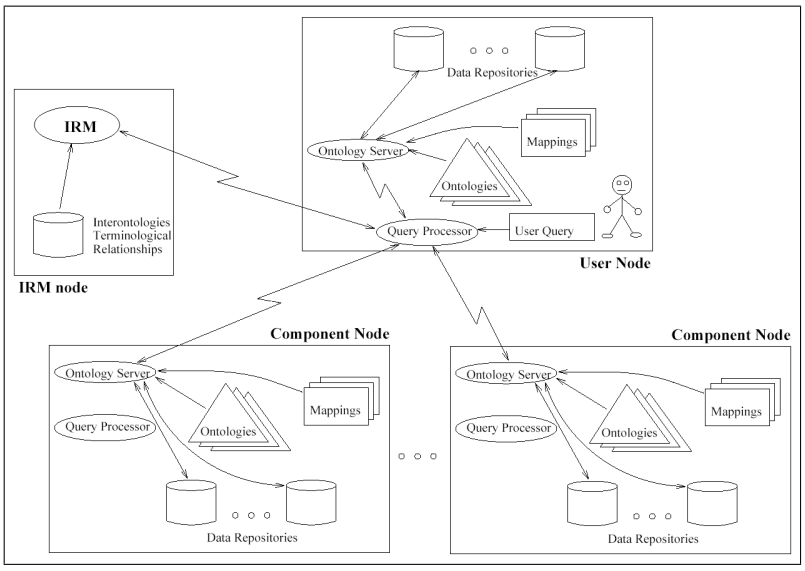

Figure 4.3: The general OBSERVER architecture

component nodes. Besides the ontology, each component node contains a number of data repositories along with mappings to the ontology, to enable semantic querying of data residing in these repositories. When other components need to be queried, the IRM provides mappings to ontologies of other component nodes in order to enable querying. The user views the data in the system through it's own local ontology, located at the user's component node.

4.1.4 Ontology mapping in the KRAFT project

The KRAFT[1] architecture is an agent-middleware architecture that proposes a set of techniques to map ontologies:

Class mapping Maps a source ontology class name to a target ontology class name.

Attribute mapping Maps the set of values of a source ontology attribute to a set of values of a target ontology attribute; or maps a source ontology attribute name to a target ontology attribute name.

Relation mapping Maps a source ontology relation name to a target ontology relation name.

Compound mapping Maps compound source ontology expressions to compound target ontology expressions.

The KRAFT architecture (Figure 4.4, taken from [Preece et al., 2001]) has three types of agents:

[1] http://www.csd.abdn.ac.uk/∼apreece/Research/KRAFT.html

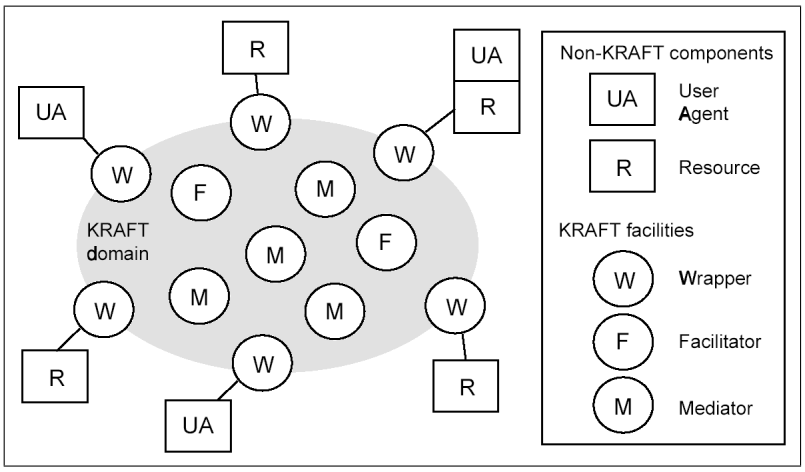

Figure 4.4: Conceptual view of the KRAFT architecture

- *Wrappers* translate the heterogeneous protocols, schemas and ontologies into the KRAFT application internal 'standards'. A wrapper agent effectively contains a one-to-one mapping between the source schema and the internal ontology.
- *Facilitators* look up services (provided by mediators and wrappers) requested by other agents.
- *Mediators* are the KRAFT-internal problem-solvers. They provide the querying, reasoning, and information gathering from the available resources. Mediators contain the mappings between the different ontologies present at the wrappers and they perform the translations between them.

The mediation between two agents in terms of matching service requesters with service providers is realized by a facilitator. It will recommend an agent that appears to provide that service. The facilitator provides a routing service by trying to match the requested service to the advertised knowledge-processing capabilities of agents with which it is acquainted. When a match is located, the facilitator informs the service-requesting agent of the identity, network location, and advertised knowledge-processing capabilities of the service provider. The service-requesting agent and service-providing agent can now communicate directly.

KRAFT defines a shared ontology in order to overcome the problem of semantic heterogeneity, among service requesters and providers. A shared ontology formally defines the terminology of the problem domain. Messages exchanged among agents in a KRAFT network must be expressed using terms that are defined in the shared ontology. Each knowledge source defines a local ontology. Then, a number of semantic mismatches will occur (homonyms and synonyms) between the local ontology and the shared ontology. To overcome these mismatches, an ontology mapping is defined for each knowledge source. An ontology mapping is a partial function that specifies mappings between terms and expressions defined in a source ontology to terms and expressions defined

in a target ontology. To enable bidirectional translation between a KRAFT network and a knowledge source, two such ontology mappings must be defined.

[Visser and Tamma, 1999] suggest using the concept of "ontology clustering" (see figure 1.11) instead of a single-shared ontology to provide heterogeneous resources integration. Ontology clusters are based on the similarities between the concepts known to different agents. The ontology clusters are organized in a hierarchical fashion, with the application ontology as the root node. The application ontology is used to describe the specific domain, which means that it is not reusable. It contains a relevant subset of WordNet concepts.

Every agent has a mapping of its local ontology to a cluster in the hierarchy. When some agents share concepts that are not shared by other agents, these new concepts are defined by creating a new ontology cluster. A new ontology cluster is a child ontology that defines certain new concepts using the concepts already contained in its parent ontology. Concepts are described in terms of attributes and inheritance relations, and are hierarchically organized.

Tool support The three different types of agents have been implemented in a prototype in the network data services area together with an industrial partner [Preece et al., 2001]. Besides this big project, several other (small) prototypes have been implemented.

Summary The KRAFT project takes an agent-based approach to information integration, where users typically have their own local ontology that is mapped to the central ontology. Three types of agents work together to provide services to the user. Wrappers provide access to data sources, mediators provide query interfaces and reasoning services, and facilitators enable the look-up of the former types of wrappers.

Originally, KRAFT used a single-shared ontology in order to enable integration of local ontologies in the overall architecture. Later on, [Visser and Tamma, 1999] suggested the use of ontology clustering for this purpose. The advantage of the use of ontology clustering is the distinction of more refined and more abstract ontologies, enabling organizing the ontology mappings in a hierarchical structure.

4.1.5 PROMPT

Noy and Musen [Noy and Musen, 1999] developed SMART which is an algorithm that provides a semi-automatic approach to ontology merging and alignment. SMART assists the ontology engineer by prompting to-do lists as well as perform consistency checking. SMART forms the basis for PROMPT [Noy and Musen, 2000b], which is an algorithm for ontology merging and alignment that is able to handle ontologies specified in OKBC [Chaudhri et al., 1998] compatible format. It starts with the identification of matching class names. Based on this initial step an iterative approach is carried out for performing automatic updates, finding resulting conflicts, and making suggestions to remove these conflicts. PROMPT is implemented as an extension to the Protégé-2000[2] knowledge acquisition tool and offers a collection of operations for merging two classes and related slots.

[2]http://protege.semanticweb.org/

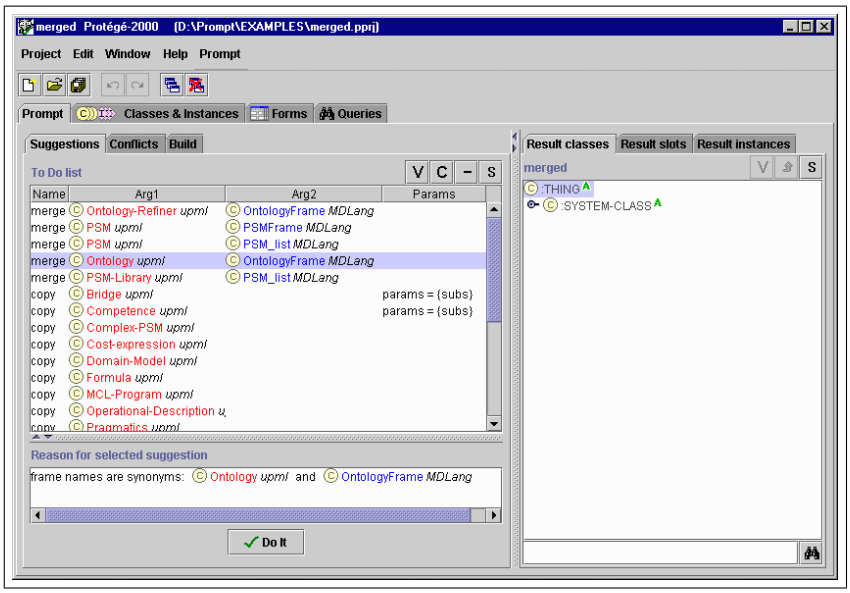

Figure 4.5: An example of Ontology Merging in PROMPT

The PROMPT algorithm consists of a number of steps. First, it identifies potential merging candidates based on class-name similarities and presents this list to the user. Then, the user picks an action from the list. The system performs the requested action and automatically executes additional changes derived from the action. It then makes a new list of suggested actions for the user based on the new structure of the ontology, determines conflicts introduced by the last action, finds possible solutions to these conflicts and displays these to the user. Figure 4.5 shows an initial to do list with merge operations. The user has selected one of the operations and PROMPT provides the reason for the suggestion ('frame names are synonyms').

PROMPT uses a measure of linguistic similarity among concept names to solve terms matching. In the first implementation of the algorithm linguistic-similarity matches were used for the initial comparison, now they concentrate on finding clues based on the structure of the ontology and user's actions.

PROMPT identifies a set of knowledge-base operations (merge classes, merge slots, merge bindings between a slot and a class, etc) for ontology merging or alignment, for each operation in this set, PROMPT defines (1) changes performed automatically, (2) new suggestions presented to the user, and (3) conflicts that the operation may introduce and that the user needs to resolve. When the user invokes an operation, PROMPT creates members of these three sets based on the arguments to the specific invocation of the operation. Among the conflicts that may appear in the merged ontology as the result of these operations are counted:

- *name conflicts* (more than one frame with the same name),
- *dangling references* (a frame refers to another frame that does not exist),
- *redundancy* in the class hierarchy (more than one path from a class to a parent other than root),

- *slot-value restrictions* that violate class inheritance.

Both the list of operations and conflicts grow as more experience is gained.

The creators of PROMPT have created an algorithm, called Anchor-PROMPT [Noy and Musen, 2000a], that enhances the detection of matching terms using non-local context. Whereas PROMPT takes only local structural similarities between terms into account, Anchor-PROMPT uses paths of a greater length in order to determine similarities. The input of the algorithm consists of a number of anchors, which are pairs of matching terms in the source ontologies. Anchor-PROMPT now takes these anchors to produce a new set of semantically close terms. It does this by comparing paths between the anchors in both ontologies. A graph is constructed with the classes as nodes and the slots as edges and a path consists of a number of edges in a graph. The Anchor-PROMPT algorithm can be used in the context of any ontology merging or aligning method and is not specific to PROMPT.

[Noy and Musen, 2003] suggest that several strategies developed for the comparison of ontology versions can also be used for finding similarities across different ontologies. While ontology versioning is concerned with finding differences between versions of an ontology, ontology aligning is concerned with its complement, namely finding similarities between different ontologies. This leads to the possibility of reuse of several matchers developed for PromptDiff [Noy and Musen, 2003] in the area of ontology merging and aligning.

Tool support The PROMPT algorithm has been implemented as an extension to the Protégé-2000 tool and as such can take advantage of all the ontology engineering capabilities of the tool. The tool supports import and export of the ontology in several ontology languages as well as database schemas using the appropriate JDBC connection, and therefore allows merging of both ontologies and data schemas. The plug-in architecture allows for the inclusion of different other schemas sources. Examples of plug-ins available for storage are DAML+OIL support, XML Schemas, and RDF(S).

Summary PROMPT provides a solution for ontology merging, not specifically data schema integration. In the ontology merging in PROMPT it is assumed that the original ontologies no longer exist after the merged ontology has been created. Therefore, there are no mappings between the original and the merged ontologies, as there are in most data schema integration solutions.

PROMPT uses a semi-automatic approach to the merge process. It uses linguistic similarities to determine possible candidates for merging and presents these choices to the user. Conflicts arising during merging (e.g. merged concepts that refer to concepts in one of the original ontologies) are detected and presented to the user along with possible solutions to the problem. During tests done with PROMPT it turned out that in the merging of ontologies about 74% of all operations executed had been proposed by the system [Noy and Musen, 2000b].

4.1.6 Chimæra

Chimæra [McGuinness et al., 2000] is an ontology merging and diagnostic tool developed by the Stanford University Knowledge Systems Laboratory (KSL).

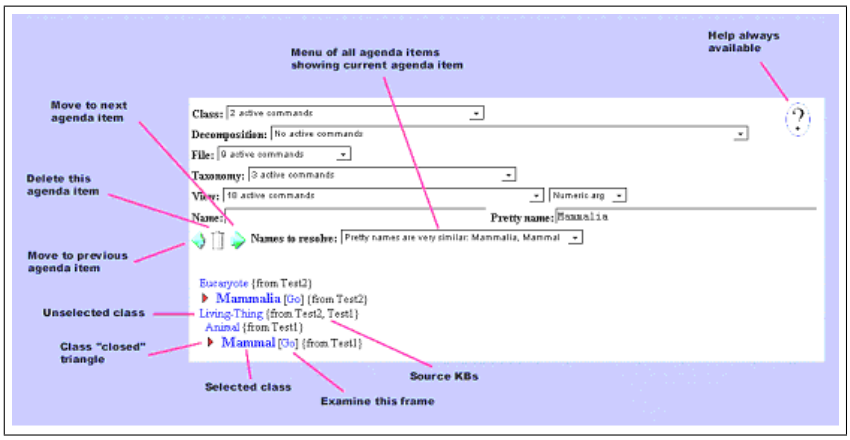

Figure 4.6: The Chimæra tool in name resolution list mode

[McGuinness et al., 2000] distinguish two major tasks in the merging of ontologies, namely (1) to coalesce equivalent terms from the source ontologies so that they are referred to by the same name in the target ontology and (2) to identify related terms in the source ontologies and identify the type of relationship (e.g. subsumption and disjointness). Chimæra support the merging task by generating two resolution lists, a name resolution list and a taxonomy resolution list.

The name resolution list (Figure 4.6) contains terms from different ontologies that are candidates to be merged or that have taxonomic relationships that have not yet been identified. Chimæra finds these suggestions based on the names of the terms, the presentation names, the definition of terms, etc The taxonomy resolution list suggests areas in the taxonomy that are candidates for reorganization. It finds such edit points by looking for classes that have direct subclasses from more than one ontology.

The name and taxonomy resolution lists correspond to two modes of operation in the Chimæra tool . In the name mode similar classes are presented that are candidates for merging. In the taxonomy mode areas of the merged taxonomy are presented that might contain conflict, such as subclasses that came from different source ontologies. Besides these two modes, there is also the slot traversal mode that guides you through the classes that have slots that came from multiple different source ontologies and might need editing.

Besides the merging of ontologies, Chimæra also supports a number of diagnostic tasks, like completeness checking, syntactic analysis, taxonomic analysis, and semantic evaluation.

Tool support The Chimæra tool has been implemented as a web application, that is available at Chimæra's web site[3]. The tool has been built on top of the OntoLingua[4] distributed collaborative ontology engineering environment, although ontologies developed in any OKBC-compliant [Chaudhri et al., 1998] application could be used in Chimæra. The editing functionality in Chimæra is

[3]http://www.ksl.stanford.edu/software/chimaera/
[4]http://www.ksl.stanford.edu/software/ontolingua/

restricted. Currently, editing and merging support is only available for classes and slots, but there are plans to include support for the merging of facets, relations, functions, individuals, and arbitrary axioms.

Summary Chimæra [McGuinness et al., 2000] is a browser-based editing, merging, and diagnosis tool. For the merging of ontologies, the system employs similarity matching between names of classes and properties in the original ontologies. Based on these similarities, the system presents a name resolution list suggesting terms that are candidates to be merged. Chimæra also employs heuristics to identify areas in the taxonomy that are candidates for reorganization.

4.1.7 ONION

ONION (ONtology compositION) [Mitra et al., 2000; Mitra and Wiederhold, 2001] is an architecture based on a sound formalism to support a scalable framework for ontology integration that uses a graph-oriented model for the representation of the ontologies.The special feature of this system is that it separated the logical inference engine from the representation model (the graph representation) of the ontologies as much as possible. This allowed the accommodation of different inference engines in the architecture.

In ONION there are two types of ontologies, individual ontologies, referred to as *source ontologies* and *articulation ontologies*, which contain the terms and relationships expressed as articulation rules (rules that provide links across domains). Articulation rules are established to enable knowledge inter-operability, and to bridge the semantic gap between heterogeneous sources. They indicate which terms individually or in conjunction, are related in the source ontology [Mitra et al., 2000]. SKAT (the Semantic Knowledge Articulation Tool) [Mitra et al., 1999] uses the structure of these graphs together with term-matching and other rules to propose articulation rules for the articulation ontologies. The source ontologies are reflected in the system by the usage of wrappers.

The mapping between ontologies is executed by ontology algebra [Wiederhold, 1994; Mitra and Wiederhold, 2001]. Such algebra consists of three operations, namely, intersection, union and difference. The objective of ontology algebra is to provide the capability for interrogating many largely semantically disjoint knowledge resources, given the ontology algebra as input. The description of the algebra operators is as follows:

- The *intersection* produces an ontology graph, which is the intersection of the two source ontologies with respect to a set of articulation rules, generated by an articulation generator function. The nodes in the intersection ontology are those that appear in the articulation rules. The edges are those edges between nodes in the intersection ontology that appear in the source ontologies or have been established as an articulation rule. The intersection determines the portions of knowledge bases that deal with similar concepts.

- The *union* operator generates a unified ontology graph comprising of the two original ontology graphs connected by the articulation. The union presents a coherent, connected and semantically sound unified ontology.

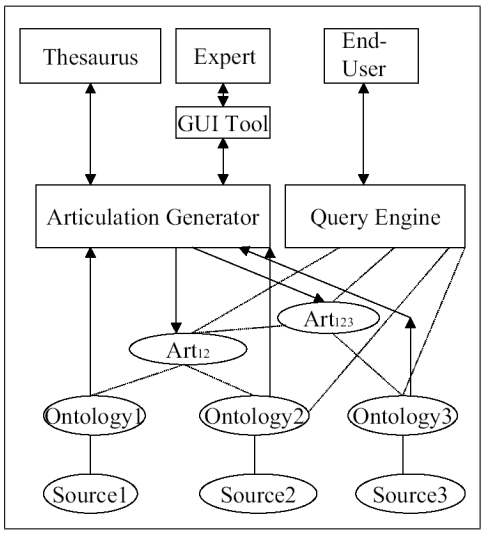

Figure 4.7: The components of the ONION system

- The *difference* operator, to distinguish the difference of two ontologies ($O_1 - O_2$) is defined as the terms and relationships of the first ontology that have not been determined to exist in the second. This operation allows a local ontology maintainer to determine the extent of one's ontology that remains independent of the articulation with other domain ontologies so that it can be independently manipulated without having to update any articulation.

ONION tries to separate as much as possible the logical inference engine from the representation model of the ontologies, allowing the accommodation of different inference engines. It also uses articulations of ontologies to interoperate among ontologies. These articulation ontologies can be organized in a hierarchical fashion. For example, an articulation ontology can be created for two other articulation ontologies that unify different source ontologies. The ontology mapping is based on the graph mapping, and at the same time, domain experts can define a variety of fuzzy matching.

Tool support The ONION architecture [Mitra et al., 2000; Mitra and Wiederhold, 2001] (figure 4.7, taken from [Mitra and Wiederhold, 2001]) consists of four components:

- *The ONION data layer.* This layer contains the wrappers for the external sources and the articulation ontologies that form the semantic bridges between the sources.
- *The ONION viewer.* This is the user interface component of the system. The viewer visualizes both the source and the articulation ontologies.
- *The ONION query system.* The query system translates queries formulated in term of an articulation ontology into a query execution plan and executes the query.

- *The Articulation Engine.* The articulation generator takes articulation rules proposed by SKAT [Mitra et al., 1999], the Semantic Knowledge Articulation Tool, and generates sets of articulation rules, which are forwarded to the expert for confirmation.

Summary ONION takes a centralized, hierarchical approach to ontology mapping, where the user views the (global) articulation ontologies. The source ontologies are mapped to each other via articulation ontologies that are in turn used by the user to express queries. The articulation ontologies are organized in a tree structure. An articulation ontology used for the mapping of two source ontologies can in turn be one of the sources for another articulation ontology (e.g. in figure 4.7 Art_{12} is one of the sources of Art_{123}). The creation of a hierarchy can be seen as a form of ontology clustering. But while [Visser and Tamma, 1999] take a top-down approach (first the root application ontology is specified, then child ontologies are created as is necessary), ONION takes a bottom-up approach in the creation of the articulation ontologies; furthermore, there is no defined root ontology for the cluster.

4.2 Comparison of the Methods

In this section, we will provide a comparison of all the methods and tools presented in the previous section and the method and tool used in the COG project. First, we will outline the criteria used for comparison of the methods and tools, then we will provide the comparison according to these criteria.

4.2.1 Comparison criteria

Some methods and tools have not been made especially for data schema integration, so the applicability to this area will be one of the criteria.

Another criterium is the maturity of the tool(s). Some tools are just academic prototypes, other tools have reached industrial strength. This is very relevant, because the heterogeneity of data schemas is a very real problem in industry today.

- *Integration paradigm*: merging or aligning or a combination (i.e. central ontology while retaining the source ontologies and mappings between the sources and the central ontology).
- *Mapping pattern* (1-to-1, single-shared, or ontology clustering). Does the methodology support one-to-one ontology mapping, single-shared ontology or ontology clustering?
- *User model*: local or global. In the case of a local model, the user views the system through his/her own local data model. In the case of a global model, every user sees the system through the global model that has been defined to encompass all data sources in the system. This corresponds to the run-time dimension identified in 1.3.
- *Mapping support.* Types of mappings that are supported. We distinguish the following types here:

  - Class mappings
  - Property (i.e. relation) mappings
  - Instances
  - Axioms / rules / constraints
  - Value transformations (for properties)
  - Conditional mapping

- Degree of *automation*. Is the mapping process manual, semi-automatic (i.e. interactive) or automatic?
- *Information sources* used for discovering concepts/similarities. It is assumed that always the source models (i.e. data schemas and/or ontologies) are used as a source of information and that it is always possible to get information from the domain expert. Possible additional information sources are existing lexicons, such as WordNet.
- *Inter-operability* with other tools. This concerns mainly the import and export languages supported by the tool.
- *Visualization* support. In what way are the ontolog(y)(ies) (and the mappings) visualized?
- *Evaluating* ontology. In what way is the resulting ontology and/or mapping evaluated?
- *Version* and maturity of the tool. There is in general a difference in maturity (and thus quality) of tools that have been in use in industry for some time and academic prototypes. The version listed is the version available to the author at the time of writing.
- *Experience* using the tool in different projects. The experiences gained with a tool can tell us a lot about the usability and the usefulness of a tool.

We compare the methodologies using these criteria in the next section.

4.2.2 Comparing the methodologies for Semantic Schema integration

Table 4.1 shows us the comparison of the tools and methods mentioned in section 4.1 and the method employed in the COG project (as described in chapter 2).

The integration paradigm for an approach influences the possible mapping patterns. For example, a pure ontology merging tool cannot be classified using any mapping paradigm, because the source ontologies do not remain after the merge process and thus no mappings are created. In the case of a pure aligning solution (e.g. OBSERVER), only a one-to-one mapping between ontologies is possible. The approaches that have been classified as using the combination between merging and aligning either use the single-shared ontology or the ontology clustering mapping pattern. Note that the single-shared ontology approach mentioned here is never the pure single-shared ontology approach (which would mean that just the shared ontology remains and all applications must use it);

Table 4.1: Comparison of semantic data schema integration tools

	COG / Unicorn	MOMIS	Info-Sleuth	OBSERVER	KRAFT	PROMPT	Chimæra	ONION
Integration paradigm	combination	combination	combination	aligning	combination	merging	merging	combination
Mapping pattern	single-shared ontology	single-shared ontology	single-shared & one-to-one	one-to-one	clustering	-	-	clustering
User model	global	global	global	local	local	-	-	global
Mapping support	class; property; value transformations; conditional mapping	class; property; constraints	n/a	class; property; value transformations; conditional mapping	class; property; constraints; instances	-	-	class; property
Degree of automation	manual	semi-automatic	semi-automatic	manual	semi-automatic	semi-automatic	semi-automatic	semi-automatic
Interoperability	import of RDBMS, XML Schema, COBOL Copybook, custom wrappers; (limited) export of DAML+OIL	custom wrappers	import: java templates for JDBC, text, etc. interoperation with OKBC	custom wrappers	IDL; XML Schema; custom wrappers	Any language supported by Protégé	Any language supported by OntoLingua	n/a

continued on next page

continued from previous page

	COG / Unicorn	MOMIS	Info-Sleuth	OBSER-VER	KRAFT	PROMPT	Chimæra	ONION
Additional Information sources	-	WordNet	-	-	WordNet	-	-	-
Visualization support	ontology and mappings	-	-	-	n/a	ontology using Protégé	n/a	n/a
Evaluating ontology	manual using test instances and database queries	manual	n/a	n/a	n/a	manual	semi-automatic using diagnostic tests	n/a
Version	2.5.1	academic prototype	academic prototype	academic prototype	academic prototype	2.0.4	0.1.43	academic prototype
Experience in projects	COG project & several pilot projects	a few test projects	EDEN	bibliographic data	network data services	applied in many projects	HPKB project & several test projects	prototype vehicle manufacturing and transportation

instead, in each of the approaches that has been classified as a single-shared ontology approach, the source ontologies remain along with a mapping to the shared ontology.

PROMPT and Chimæra could not be classified along the run-time dimension, because both are just merging tools. They produce a merged ontology built from the source ontologies, but they have no run-time architecture.

A note must be made on the degree of automation classification. Every method/tool has been classified as either manual or semi-automatic. There is, however, between the semi-automatic integration approaches much difference in the amount of automation. Please review the corresponding descriptions in section 4.1 if more information about the automation is needed.

The inter-operability with other tools is measured in the supported input and output platforms/languages. Note that there is a difference in import for data schema integration and ontology integration tools. The former usually require mappings to the database platform, while the latter usually only requires support for the import of several ontology languages. Most approaches support the development of custom wrappers for data schemas or custom import and export functionalities for ontology languages.

It is especially important to evaluate the maturity of a tool if the tool is to be used in an industrial setting, as was the case with COG. The tool used in the COG project, the Unicorn Workbench, is still under heavy development, but has reached a certain maturity. Of the other approaches we have evaluated here, only PROMPT can claim a maturity that exceeds the phase of academic prototype. Measures for the maturity of a tools are the version and the experiences with the tool in projects.

In the next chapter, we will bring the information integration problem to a new level, where the information integration problem is expanded to an inter-organizational level. We will introduce the vision of the Semantic Web and the problem of ontology mapping on the Semantic Web.

Chapter 5

Enabling Semantic Interoperation on the Semantic Web

This chapter is partly based on deliverable D14 v1.0 of the Esperonto project[1], with the title 'Ontology Alignment Solution' [de Bruijn et al., 2003b].

In chapter 2 we have seen how, in the context of the COG project, data residing in different databases and other data sources, such as XML and Excel, can be semantically integrated using ontology technology. In chapter 3 we showed a possible application of such an integrated architecture by showing how the user can, through the semantic middle-ware, issue queries on the actual databases while retaining a global view, only having to deal with the conceptual view provided by the global ontology.

Now, the problem gets harder when different organizations want to interoperate. In a single organization it is to some extent possible to "force" people to use a single conceptual data model, or ontology. When a single terminology is to be used by different organizations, it is very hard to agree on this single terminology, and extra translations need to be created from the internal data representation of a single organization and the external representation.

When organizations have already described their data in a formal explicit way, using ontologies, this problem is reduced in complexity. Because the meaning of an ontology is entirely captured in its structure and formal semantics, it is easier for a different entity to understand this ontology and, more importantly from an automation point-of-view, for a machine to understand it.

Unfortunately, if there is no formal relationship between the external ontology and the internal conceptual model, an agent still can not "understand" the ontology. There needs to be a formal relationship between the two ontologies, so that the agent can relate the concepts in the external ontology with known concepts in the internal ontology.

In this chapter we describe the ontology mapping problem we encounter in the Esperonto project and propose a solution to ontology mapping.

[1] http://esperonto.semanticweb.org/

This chapter is organized as follows: in the next section we describe the generic ontology mapping problem through an example, after which we introduce the Esperonto project and the context of the ontology mapping problem in the project. We identify problems that arise in relating ontologies in section 5.3 and perform a requirements analysis on the solution in section 5.4. We then propose a solutions for the ontology mapping problem in the Esperonto project in section 5.5, taking into account existing algorithms and tools, and conclude with some limitations of the proposed approach and identify future work in section 5.6.

5.1 Ontology Mapping

A way to specify the relationships between two different ontologies is to create an explicit ontology mapping. Such an ontology mapping consists of a set of mapping rules, which relate the concepts in the different ontologies with each other. Such a mapping is illustrated in Figure 5.1.

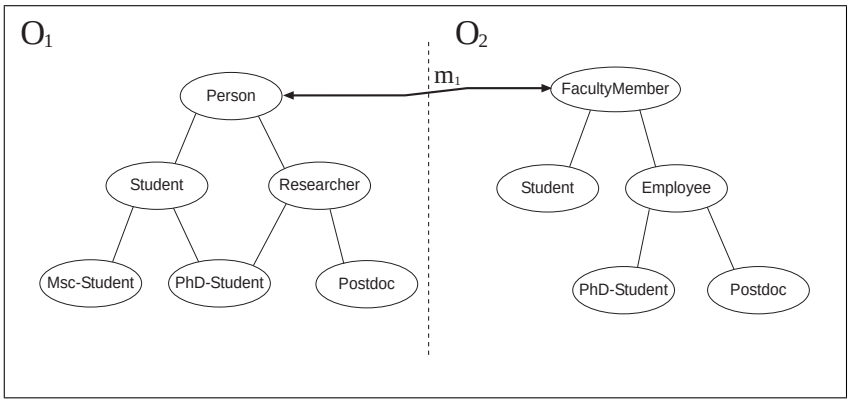

Figure 5.1: Example ontology mapping (1)

In the example, all thin lines represent is-a relations. This means, in ontology $\mathcal{O}_1$ a *Student* is a $Person$, a *Researcher* is a $Person$, etc ...

To begin with, the ontologies are unrelated. We specify an explicit mapping, m_1 between the concept $Person$ in ontology $\mathcal{O}_1$ and the concept $FacultyMember$ in ontology $\mathcal{O}_2$. Thereby, we explicitly specify a relationship between the ontologies. If mapping rule m_1 states equivalence of the concepts $Person$ and $FacultyMember$, this means that every human according to $\mathcal{O}_1$ is also a faculty member according to $\mathcal{O}_2$, and vice versa.

Now that we have created this one mapping rule, m_1, we can deduce additional facts from m_1 combined with the relationships already present in the ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$. For example, since every *Student* is a $Person$ and every person is a $FacultyMember$, we can deduce that every *Student* must be a $FacultyMember$.

We shall from now on write the name of the ontology in front of the name of the concept. So, the concept *Student* in $\mathcal{O}_1$ is from now on identified as $\mathcal{O}_1.Student$ and *Student* in $\mathcal{O}_2$ becomes $\mathcal{O}_2.Student$.

We can say that every instance of $\mathcal{O}_2.Student$ is an instance of $\mathcal{O}_1.Person$. Note that we do not know anything about the relation between $\mathcal{O}_1.Student$ and $\mathcal{O}_2.Student$. All we now is that they are both subconcepts of $\mathcal{O}_1.Person$ and $\mathcal{O}_2.FacultyMember$ (because of the equivalence relation m_1). It might be possible that there exists no instance of $\mathcal{O}_1.Student$ that is also an instance of $\mathcal{O}_2.Student$ or it might be possible that every instance of $\mathcal{O}_1.Student$ is also an instance of $\mathcal{O}_2.Student$. Note that we cannot rule out the latter option, because there might be additional facts that are currently not known, which would make these concepts equivalent. There might be a relation somewhere on the web that somehow relates these two concepts. We can illustrate this with the mapping rule m_2 in Figure 5.2.

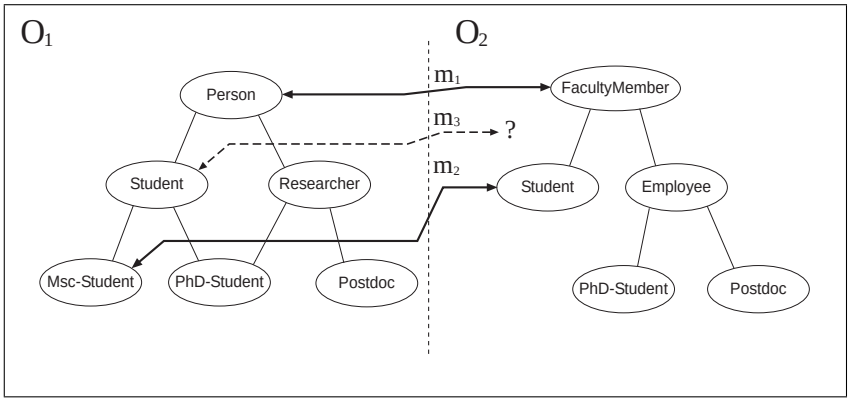

Figure 5.2: Example ontology mapping (2)

Mapping rule m_2 states equivalence between the concepts $\mathcal{O}_1.Msc-Student$ and $\mathcal{O}_2.Student$. We now know more than in the situation of Figure 5.1. We know that the concepts $m_1.Msc - Student$ and $m_2.Student$ are equivalent. Note that now actually $\mathcal{O}_2.Student$ is a subconcept of $\mathcal{O}_1.Student$. This means that the set of instances of $\mathcal{O}_2.Student$ is a subset of the set of instances of $\mathcal{O}_1.Student$. If we now want to map the concept $\mathcal{O}_1.Student$ to a concept in $\mathcal{O}_2$, we have a problem. In generality, $\mathcal{O}_1.Student$ falls somewhere between $\mathcal{O}_2.FacultyMember$ and $\mathcal{O}_2.Student$, but there is no concept we have directly mapped it to, as illustrated by m_3 in Figure 5.2.

We have introduced ontology mapping through the use of an example. We will now describe the context of the ontology mapping problem in the Esperonto project and, more generally, the Semantic Web.

5.2 Ontology Mapping on the the Semantic Web

In this section, we first present the general view on the Semantic Web, after which we describe the role of the Esperonto project in the Semantic Web, and position the Ontology Mapping problem in the Esperonto project.

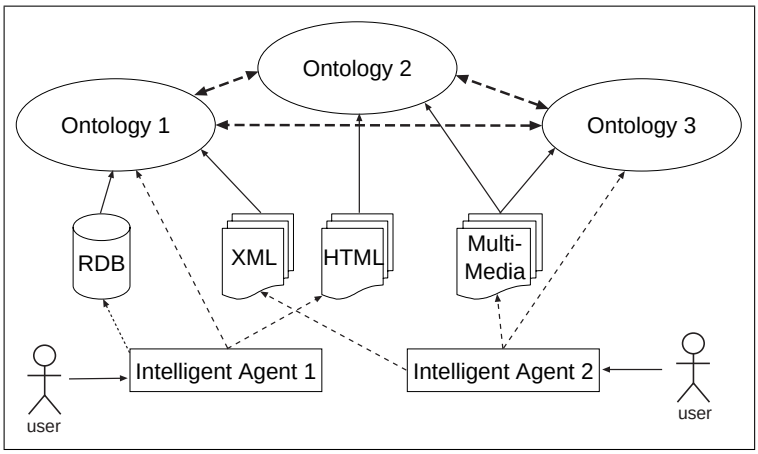

Figure 5.3: Simple depiction of the Semantic Web

5.2.1 The Semantic Web

Tim Berners-Lee, known as the inventor of the World Wide Web (WWW), has a vision for the future of the World Wide Web, which he calls "The Semantic Web" [Berners-Lee et al., 2001]. In this Semantic Web, information will be presented in machine readable form. Right now most information present on the WWW is presented in natural language, only understandable to humans. And although there have been some advancements in the field of text-recognition, there are still a lot of issues to be resolved before natural language can be understood by computers [Fensel, 2003].

This means, a new way of recording the information has to be found in order to make the information on the Web understandable by machines. A way to formally specify knowledge is to use ontologies, as described above. An information source on the Web would contain a reference to an ontology or some sort of annotation (which also uses some ontology), which contains the definition of the knowledge present in the information source. Intelligent agents can now gather information from different sources and combine the information, because of the formal relationships in the ontologies. Two related data sources can either use the same ontology, in which case it is straightforward for an agent to combine the information, or they can use two different ontologies, which have been related to each other using formal mappings. These mappings enable the intelligent agent to combine data in different related sources to fulfill the information requests by the user.

Figure 5.3 shows the relations between the users, the intelligent agents, the data sources and the ontologies on the Semantic Web. The data sources, the content on the Web, can vary from HTML and XML documents to multimedia content, such as pictures or movies. We furthermore also see databases as part of the Semantic Web. Currently, information in company-internal databases is only partly shared on the Web by publishing to HTML and XML documents. We envision standardized access mechanisms to be developed, which provide access to the underlying data sources on the basis of ontology annotations. The most interesting documents are, as we shall see, the documents that are exchanged between applications and between organizations. In the next section

we discuss the most important application areas of Semantic Web technologies, in the sense of impact of the area.

5.2.2 The Esperonto project and Ontology Mapping

The Esperonto project is an EU funded project (under contract number IST-2001-34373) in the Information Society Technologies (IST) Program for Research, Technology Development and Demonstration under the 5th Framework Program of the European Commission. The project runs from 2002 to 2005.

The main goal of the Esperonto project is to bring content on the current World Wide Web (WWW) to the Semantic Web [Berners-Lee et al., 2001], thereby bridging the gap between the current Web and the Semantic Web. To achieve this goal, two main objectives were formulated. The first is to "**construct a service that provides content providers with tools and techniques to publish their** (existing and new) **content on the SW, independently of their native language**", the second is "**providing added value knowledge-based services on top of the constructed Semantic Web**."

The first objective is concerned with adding annotation to existing documents on the Semantic Web; both textual and multimedia documents. Besides that, also data residing in databases is to be lifted to the Semantic Web, by mapping these schemas to ontologies. We have sufficiently covered this area in chapter 2 in this thesis, so we will not go into this aspect here.

We are concerned here with the second objective, "providing added value knowledge-based services on top of the constructed Semantic Web", and especially with services enabling inter-operation between different entities on the Semantic Web.

An entity on the Semantic Web can be any organization, group of people, or even single person on the Semantic Web. Such an entity on the Semantic Web is typically represented by an agent that performs tasks on the Semantic Web on behalf of the entity. This view on the Semantic Web, consisting of documents with their annotations, ontologies and agents representing entities is illustrated in Figure 5.4.

The following layers (starting from the bottom) are distinguished in this view on the Semantic Web:

Instance layer In this layer, we have the actual data residing on the Web. Examples of data formats are video, pictures, natural language texts in html documents, semi-structured XML documents and structured databases.

Annotation layer Annotations link instance data with ontologies. Annotation add formal semantics to the instances, making the instance data machine processable. Note than an annotation can link an instance with more than one ontology (as is the case with the second left and the right annotation in Figure 5.4).

Ontology layer This layer consists of the ontologies on the Semantic Web and their *formal relationships*, expressed through *explicit mappings*. Because of these explicit mappings, an agent, which knows about one ontology, can process instance data annotated with another ontology if there exists a mapping.

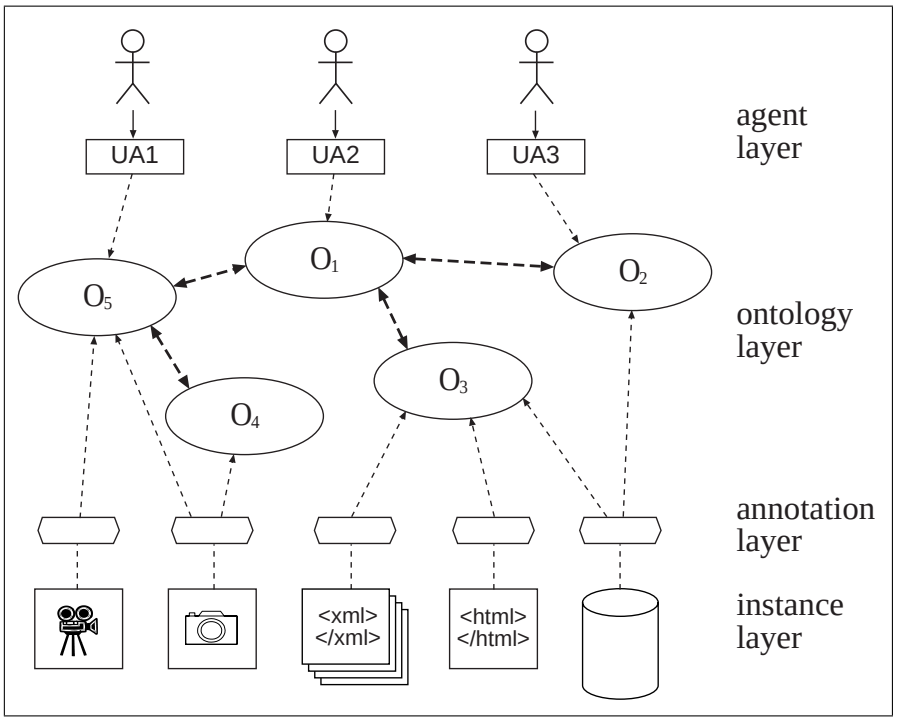

Figure 5.4: Instance, annotation, ontology and agent view of the Semantic Web

Agent layer The agent layer consists of the agents that actually operate on the Semantic Web. Agents perform user tasks on the Semantic Web. They use instance data, together with ontologies, to solve complex problems for the user. Furthermore, agents can communicate with other agents in order to solve these tasks. We will not describe the latter case here, but we do discuss upcoming technologies around Semantic Web Services [McIlraith et al., 2001; Fensel and Bussler, 2002; Lara et al., 2003] in the outlook in chapter 6.

The Esperonto project is mainly concerned with the annotation layer and the ontology layer. For this purpose, a software prototype is being developed in the project, called the Semantic Annotation Service Provider (SemASP). SemASP provides annotation using ontologies of content residing on the Web and in databases. There is, however, also an ontology management module, which takes care of ontology engineering (creating ontologies), ontology maintenance and *ontology mapping*.

We concern ourselves here with the aspect of ontology mapping (cf. the wide dashed double arrows in Figure 5.4) as it is to be applied in the SemASP prototype system.

5.3 Problems in Ontology Mapping and Aligning

[Klein, 2001] identifies two levels of mismatches between ontologies. The first level is the ontology language or meta-model level. These mismatches include syntactic mismatches, differences in the meaning of primitives in the different languages, and expressivity of the languages. We assume that the Ontology Management module of SemASP, through its import and export functionalities, will handle these differences and present the ontologies to the alignment module in OKBC format. We therefore do not have to worry about mismatches at the language level.

The second level of mismatches is the ontology or model level. Where mismatches on the language level include differences in encoding and meaning of language constructs, mismatches at the ontology level include mismatches in the meaning or encoding of concepts in different ontologies. These mismatches are exactly the problems we are facing for the ontology alignment solution. [Klein, 2001] follows the basic types of ontology mismatches identified in [Visser et al., 1997]:

- *Conceptualization mismatches* are mismatches between different conceptualizations of the same domain.
- *Explication mismatches* are mismatches in the way a conceptualization is specified.

Klein distinguishes two different conceptualization mismatches:

- *Scope mismatch.* Two classes have some overlap in the extension (the set of instances), but the extensions are not exactly the same. [Visser et al., 1997] call this a class mismatch and work it out further for classes and relations.
- *Model coverage and granularity.* This mismatch is a difference in the part of the domain that is covered by both ontologies or the level of detail with which the model is covered.

Klein furthermore distinguishes different types of explication mismatches. First, there are two mismatches in the style of modelling:

- *Paradigm.* These mismatches occur when different paradigms are used for the explication of the same concept. For example, one ontology might represent time using intervals, while another ontology might use points to represent time.
- *Concept description.* Mismatches in the way a concept is described. For example, differences in the way the is-a hierarchy is built or when in one ontology several sub-concepts are defined for groups of instances, while in the other ontology subclasses are created for these different groups.

Then there are the terminological mismatches:

- *Synonym terms.* Two terms are equivalent when they are semantically equivalent, but are represented by different names. It is possible to use dictionaries or thesauri to resolve this problem, but one should be aware of possible scope differences (see the first conceptualization mismatch above).
- *Homonym terms.* This problem occurs when semantically different concepts have the same name.

Finally, the last type of difference:

- *Encoding.* Values in different ontologies might be encoded in a different way. For example, one ontology might define distance in kilometers, while another uses miles.

[Mena et al., 2000] have identified different types of inter-ontology relationships (based on relationships identified in [Hammer and McLeod, 1993]) that should be taken into account by ontology mapping and alignment systems:

Synonym Two terms in different ontologies have the same semantics. This corresponds to the synonym terms mismatch mentioned above.

Hyponym A term is less general than another term in a different ontology. This is a special kind of scope mismatch and can also be seen as a concept description mismatch.

Hypernym A term is more general than another term in a different ontology. This is a special kind of scope mismatch and can also be seen as a concept description mismatch.

Overlap There is an intersection in the abstraction represented by two terms. This corresponds to the scope mismatch.

Disjoint There is no intersection in the abstraction represented by two terms.

Covering The abstraction represented by a term in one ontology is the same as the abstraction represented by the union of other given abstractions which are subsumed individually by the term. This corresponds to the granularity mismatch identified by Klein.

In the next section, we will use these problems identified by [Klein, 2001] and [Mena et al., 2000] in order to identify requirements on the ontology alignment solution.

5.4 Requirements Analysis

We identify requirements for an ontology alignment solution along two dimensions. First, we identify different types of mappings that should be supported by the tool with regard to the problems defined above and possible automation in finding these mappings. Then, we identify user requirements and requirements for the User Interface for the ontology alignment functionality in SemASP.

5.4.1 Mapping Language Requirements

Using the problems in inter-ontology relationships identified by [Klein, 2001] and [Mena et al., 2000] and mentioned in the previous section, and our own experience with ontology mapping, we have identified types of mappings that need to be provided by an ontology mapping and aligning tool, in order to deal with these problems:

- *Equivalence* between terms (concepts and properties). When two concepts in different ontologies are conceptually the same, an equivalence relationship should be established. This solves the problem of *synonym terms*.
- *Sub- and superclass relationships*. We need to be able to specify that a concept in one ontology is more or less general than a concept in another ontology. These relationships can be used to resolve the *hyponym* and *hypernym* problems. Actually, these sub- and superclass relationships are special kinds of partial mappings.
- *Partial mappings*. When there is overlap between terms in different ontologies, but this overlap is incomplete (i.e. the terms are not equivalent), a partial mapping needs to be created stating which parts of the extensions of the concepts is equal. This can be done by describing the intersection of the concepts using rules. This covers the problem of *scope mismatch* and a part of the *concept description mismatch* and *granularity mismatch*. Note that for certain types of mismatches the intersection encompasses the entire extension of one of the concepts, in which case the partial mapping could be interpreted as a subclass relationship, even though this is not always the case (e.g. for *homonym terms*); when the intersection encompasses the entire extension of both concepts, this mapping is simply the equivalence relationship.
- *Value transformations*. A weight might be represented in kilograms in one ontology and in grams in the other. When mapping these weight concepts, we need a transformation function, which transforms kilograms into grams and vice versa. These functions can be used to solve *encoding mismatches*.
- *Union*. In order to solve the problem of *covering terms* and *mismatches in granularity*, it is necessary to be able to specify that a term in an ontology is the union of several terms in another ontology.
- Finally, we need to be able to specify *Disjointness* of terms in order to mitigate the problem of *disjoint terms*, i.e. when the intersection of the extensions of the terms is empty.

We still want to retain the equivalence and subclass relationships, even though these are special cases of partial mappings, because these relationships are better understandable to the human user and their discovery is easier to automate than general partial mappings. Equivalence and subclass relationships can in general be discovered using a thesaurus, such as EuroWordNet[2], which contains synonym, hypernym, and hyponym relationships for words in different languages, or by using structural analysis, as done for example in

[2]http://www.illc.uva.nl/EuroWordNet/

Anchor-PROMPT [Noy and Musen, 2000a]. Note that it is not possible to completely automate the creation of equivalence and subclass relationships, because of a possible *scope mismatch*, as was noted in the previous section; some user interaction is still required.

5.4.2 Possibilities in automating the creation of mappings

At this point in time the automatic mapping of ontologies seems more science fiction than reality. Therefore, we focus on semi-automatic mapping, where the system will suggest mappings between concepts in the source ontologies and the user will either discard or follow these suggestions.

5.4.3 User Interface Requirements

Because the mapping process cannot be completely automated, we need a User Interface to allow user interaction with the alignment module. We assume for now that the user is presented with suggestions for possible mappings that have been found by the system. The user can choose to either follow or discard the suggestions, on a per suggestion basis (i.e. after the user chooses to follow a suggestion, the list of suggestions is revised by the system). For this we need mechanisms for the display of the suggestions to the user and for the user to provide feedback on these suggestions.

We will furthermore assume that the system will not be able to find all required mapping rules. The user has to be able to manually specify a mapping rule. We need an advanced User Interface for this, where it is possible for the user to select a certain mapping type and to select concepts from the ontologies that are to be mapped using this mapping rule. Furthermore, the user needs to be able to specify certain parameters for the mapping rules, which is necessary for the partial mappings and the value transformations. For the value transformations we need an expression language that is able to perform transformations on string and numeric values, which can also be used by the user as a parameter for this mapping rule. For the partial mappings the user needs to be able to create arbitrary formulas using the concepts in the ontologies in order to describe the intersection of the concepts to be mapped.

The following functions will be offered to the user:

Choose two ontologies to align Before the ontology alignment process can start, the user needs to select two ontologies O_1 and O_2 in the ontology management module and give the command to start the alignment.

Choose an existing alignment to edit The user needs to be able to modify existing alignment in the case of faulty or missing mappings between concepts.

Provide feedback on generated suggestions When the alignment process has started, the system will try to find suggestions for mappings between concepts in O_1 and O_2 and present these to the user. The user can now choose to either follow or discard these suggestions (on a per-suggestion basis).

Create a mapping rule When a user identifies a mapping rule to be applied that is not on the list of suggestions provided by the system, the user will

be able to select the type of mapping rule to be defined, the concepts in O_1 and O_2 to apply the rule to, and, if necessary, some parameters (i.e. for the partial mapping and the value transformation mapping rules).

Store mapping Once the user has decided that all necessary mapping rules are in place, the user will tell the system to store the defined mapping for later use.

These User Interface requirements provide an overview of the functionality to be offered to the user. The solution defined in section 5.5 accommodates these user requirements, as well as the mapping language requirements put forward above.

5.5 A Solution for Ontology Mapping

In this section we define our ontology alignment solution following the main principles outlined in the introduction to this chapter and the problems identified in section 5.3, taking into account the experiences gained with existing tools, as described in the previous section.

We have identified four main principles on which the solution should be based (we provide with each principle the argumentation to support our choice):

One-to-one mapping For each ontology a set of translating functions is provided to allow the communication with another ontology without using an intermediate ontology.

Because of the distributed nature of the Web, the management of ontologies can not be centralized. Individual groups need to be able to update their own local data models, in order to enable flexibility. Because we want to make sure that a group on the Semantic Web is responsible for its own ontology, we propose to use one-to-one mapping of ontologies. In the case of single-shared ontologies or ontology clustering, groups would have to hand over the responsibility for their own ontology to some central organization, which is undesirable for such an inherently distributed and heterogeneous group as the entities on the Web.

Drawback of one-to-one mapping is that potentially many mappings need to be created between all the different local ontologies. This could result in $O(n)$ mappings for n ontologies. When, however, mappings are specified in a declarative, symmetric (a two-way mapping instead of a one-way mapping) way using a standardized mapping language, the mappings could be used transitively[3]. Now, ideally, only $(n-1)$ mappings are required to map all ontologies in the given domain to each other (as in Figure 5.4). However, a mapping between two different ontologies only provides the intersection of the two ontologies, because two ontologies typically do not cover exactly the same domain and often do not have the same granularity.

Certain standard ontologies could arise for the domain, to which all ontologies in a certain field are mapped using a one-to-one mapping between

[3]As an illustration: in Figure 5.4, agent UA1 can understand ontology $\mathcal{O}_2$ because of the existence of the mappings between $\mathcal{O}_5$ and $\mathcal{O}_1$ and the mapping between $\mathcal{O}_1$ and $\mathcal{O}_2$

each local ontology and the standard ontology, which would also reduce the complexity of the mapping problem.

Semi-automatic mapping The mapping task should be performed semi-auto-matically. The system should provide mapping candidates to the user; the user will then either use these suggestions or discard them. There should also be support for manually creating mappings that have not been identified by the system.

It is desirable to have automation support in the creation of ontology mappings. It was seen, however, in the survey conducted in chapter 2, that complete automation in creating ontology mappings is not possible. Therefore, we will restrict ourselves to semi-automatic mapping.

Integration in the SemASP Ontology Management module The ontology alignment solution is to be implemented as a module in the SemASP Ontology Management module. The ontologies to be aligned are to be provided to the module by the Ontology Management module, and the created mapping is returned to the Ontology Management module, so that the mappings can be put into operation. This integration also guarantees language independence of the ontology alignment process. The import from and export to specific languages will be handled by the management module.

OKBC Compliant The internal workings of the solution should be based on the knowledge model of OKBC [Chaudhri et al., 1998] in order to ensure inter-operability with other knowledge-based applications. The principle of OKBC compliance does not contradict the language independence principle, since the language independence principle is related to the import and export capabilities of the ontology engineering environment, while the OKBC compliance refers to the internal workings of the module.

Communication with the Ontology Management module will be done using the OKBC standard for knowledge exchange. Furthermore, the OKBC knowledge model has proven itself in several ontology matching algorithms (e.g. the matching algorithms in PROMPT [Noy and Musen, 2000b] and Chimra [McGuinness et al., 2000]).Therefore, the internal workings should be based on the OKBC knowledge model.

We propose a solution for the ontology alignment problem in the Esperonto project following the principles stated above, taking into account the requirements stated in the previous sections and taking into account existing tools and algorithms, as stated above. We will first look into the reuse of existing tools and then describe the ontology mapping algorithm and the way the algorithm is to be integrated into the SemASP platform.

5.5.1 Reusing existing methods and tools

A lot of effort has already been put in the development of tools supporting ontology mapping and aligning. In section 4.1 we have analyzed existing methods and tools currently available for ontology mapping and aligning. In this section we will try to identify possible candidates for reuse in Esperonto among these methods and tools.

Since the ontology alignment component in SemASP only needs to support the alignment task (only the creation of the mapping) and does not need to support an operational information architecture, we will focus on the support for ontology alignment in the methods and tools analyzed in section 4.1.

Two algorithms that have a clear tool support and a clear algorithm for identifying mapping candidates are Chimæra [McGuinness et al., 2000] and PROMPT [Noy and Musen, 2000b]. Both tools work with the principle of to-do lists (called the name and taxonomy resolution lists in Chimæra) for the user, on which identified merging candidates are listed. The user can now tell the tool to perform actions from the to-do list, discard them, or perform custom (manual) merging actions.

A possible problem for reuse of these tools in SemASP is the fact that they are both ontology merging tools and not aligning tools. In other words, the result of the usage of these tools is an ontology unifying the concepts from the source ontologies and not a mapping between the two source ontologies. However, for the merging of ontologies it is, just like in ontology mapping and aligning, necessary to identify relationships between concepts in the source ontologies. The identification of these relationships is the same for both the merging and the aligning tasks. Therefore, it would be possible to reuse and extend the algorithms used for the identification of these relationships.

PROMPT and Chimæra seem to provide the same functionality when it comes to the identification of relationships between ontologies. However, according to [Noy and Musen, 2000b], PROMPT provides more correct suggestions than Chimæra and the suggestions provided by Chimæra are in general less specific. We have not conducted any tests to verify this claim and the author are not aware of any other comparative test of both tools.

Lets now review PROMPT and Chimæra in the light of the main principles for our solution we have identified in the introduction to this chapter:

One-to-one mapping The algorithms provided by both PROMPT and Chimæra can be used for one-to-one mapping. In fact, both tools assume two source ontologies as input and one target ontology as output. We could replace this target ontology with a list of mapping functions (which is in fact in combination with the source ontologies, a virtual merged ontology), and we would have a tool for one-to-one ontology mapping and aligning.

Semi-automatic mapping Both approaches use automatically generated to-do lists that suggest mappings of terms. Chimæra provides the name and taxonomy resolution lists and PROMPT provides a to-do and a conflicts list (containing conflicts that have arisen during the process of merging). The user uses these lists to interactively create the merged ontology. The lists contain merge candidates in the source ontologies. However, we could use these lists to identify relationships between concepts and deduce mapping rules, which can be suggested to the user.

Integration in the SemASP Ontology Management module In the original implementations, both tools are integrated in an ontology engineering environment, as PROMPT is integrated in Protg-2000 and Chimæra is based on OntoLingua. If either of these tools is to be reused,

they have to be decoupled from the original hosting environment and ported to SemASP.

OKBC Compliant Both solutions are OKBC compliant.

Now, we will look into the implementations and the possibility of reuse of these implementations. Chimæra has been implemented as a Web Application on top of OntoLingua. Chimæra is in fact only available as a Web Application running on the KSL Stanford web site and therefore not suitable for integration in our SemASP architecture, since we need full control over the module. PROMPT has been implemented as a plug-in for the Protg-2000 ontology engineering environment. These facts make it hard to reuse the existing implementations of PROMPT and Chimæra. Therefore we propose to implement the module from scratch, reusing principles from other tools and algorithms.

We can, however, reuse principles presented in PROMPT and Chimæra in order to create a solution taking into account the experience gained with previously developed tools. Also the algorithms used in both tools can be reused, along with new developments, such as Anchor-PROMPT [Noy and Musen, 2000a].

In the next section a solution for the ontology alignment problem reusing the principle of maintaining action lists for the user, as is done in PROMPT and Chimæra.

The ontology mapping algorithm

We propose to use a mixture of linguistic and structural analysis for identifying mapping candidates in the concepts of the source ontologies. The tool will maintain a mapping candidates (to-do) list as defined in PROMPT [Noy and Musen, 2000b]. The mapping candidates list will contain the possible mappings the system has found in the ontologies. The possible mapping rules correspond to the types of mapping rules identified in section 5.4.1.

The system will use class-name and slot-name similarities for detecting equivalence in classes and slots and will distinguish more- and less-general terms for detecting subsumption. Furthermore, EuroWordNet will be used for detecting synonyms (for equivalence) and hypo- and hypernyms (for subsumption).

Concepts with similar slot-names or similar relations (i.e. structural similarities) will also be proposed as candidates for mapping. Once a mapping has been created, more can be derived about all classes with relations to the mapped classes. Such related classes could also be candidates for mapping when more similarities are found. If, for example, class C_1 in ontology O_1 is mapped to class C_2 in ontology O_2 and C_3 is a class in ontology O_1, which is more general than C_1 (C_1 is-a C_3) and C_4 is a class in ontology O_2, which is more general than C_2 (C_2 is-a C_4), then C_3 and C_4 are candidates for mapping. The example is illustrated in figure 5.5. Note that if the is-a relations would be relations with an arbitrary name, the same deductions about the relationship between classes C_3 and C_4 can be made.

The user is presented with the list of suggested mappings. The user can now indicate which suggestions to follow and which ones to discard. Besides this, the user can use the editing environment to manually specify mappings between concepts in the two source ontologies.

The flow of the ontology alignment process as we have identified (based on PROMPT) consists of (see Figure 5.6):

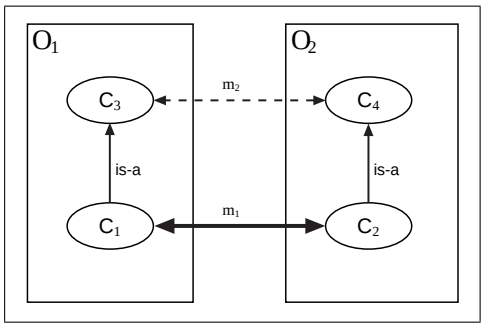

Figure 5.5: If M_1 has been specified, it is likely that there exists a link between C_3 and C_4, expressed in M_2

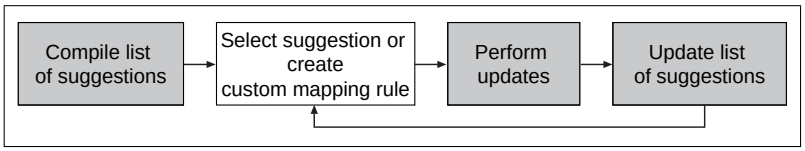

Figure 5.6: Flow of the aligning algorithm. Gray boxes indicate action by the system; the white box indicates the user action

1. *Compile list of suggestions.* Based on linguistic analysis, the system compiles an initial list of suggestions for mapping rules.

2. *Select suggestion or create custom mapping rule.* The user either selects an entry from the list of mapping rule suggestions or creates a custom mapping rule.

3. *Perform updates.* The selected or created mapping rule is added to the mapping.

4. *Update list of suggestions.* Additional (structural) analysis is executed taking into account the mapping rule(s) created by the user. If a suggestion was followed, it is taken from the list. Additional suggestions found by the system are added to the list. After this step, there is an iteration step back to the second step, where the user is presented with the updated list and can make a choice.

Integration of the Ontology Alignment module in SemASP

The Ontology Alignment module will be integrated in the Ontology Management module in SemASP. The alignment task will be initiated from the management module. The ontologies to be aligned will be input to the alignment module in OKBC-compatible format.

It should be investigated whether it is possible to integrate the front-end functionality of the alignment module with the ontology editor in SemASP. This front-end functionality could be implemented as an extension to the ontology editing functionality. Further required user interaction is the presentation of the to-do lists to the user and the user feedback to these lists. The author suggests that this functionality be also integrated in the editing environment,

so as to separate the automatic steps in the algorithm from the steps requiring user interaction.

Once the ontology mapping has been completely created, the alignment module will return the mapping to the ontology management module, where it will be stored for later use.

5.6 Conclusions and future work

In this chapter, we have looked into several existing methods and tools for ontology aligning and ontology mapping. We have identified some requirements on an ontology mapping application and the types of mappings necessary for ontology alignment.

We have defined a solution for the ontology alignment problem in the Esperonto project. We have identified some main principles to be followed by the solution, the main problems in ontology mapping and requirements on the mappings to mitigate these problems.

We have looked into the possibilities of reusing existing tools. It turned out that there was no tool that we could completely reuse. We could, however, reuse some concepts from existing algorithms, mainly PROMPT [Noy and Musen, 2000b], in our solution.

An area for further attention in a future version is conflict resolution as it is used in PROMPT. In PROMPT, a conflict list is maintained, besides the to-do list, which contains conflicts that have arisen during the merging process and possible ways to solve the conflict. In future versions of our algorithm it might be useful to identify conflicts that might arise during the alignment process.

The *is-a* relationships in figure 5.5 could be seen as paths of length one (i.e. edges) in a graph where the nodes represent classes and the edges represent slots. Anchor-PROMPT [Noy and Musen, 2000a] uses paths of greater length to detect similarities and by using multiple paths through a node to detect similarities between classes, similarity in slot names is no longer required. The Anchor-PROMPT algorithm is constructed in such a way that it can be used with any ontology merging or aligning method, so that it would be possible to plug it into this Ontology Alignment module in a future version.

Another possibility for future improvement is the use of instances of the source ontologies for detecting the mapping candidates, as is done in [Stumme and Maedche, 2001]. When evaluating instances for the purpose of detecting similarities in two different ontologies, one has to be aware of the possible limitations. In the case where many instances will have to be evaluated by the system, performance could drop significantly, which would make the mapping process too slow for frequent use. Furthermore, there might not be enough instances available to detect similarities, which could lead to detection of non-existent similarities and the failure of detection of existing similarities.

As was pointed out in [Noy and Musen, 2003], ontology alignment is one of the tasks in ontology management. Two other tasks they have identified in ontology management are **ontology versioning** and **specifying transformation rules between versions of the same ontology**. These two tasks are handled by the Ontology Evolution module in SemASP. As Noy and Musen have identified, the tasks of ontology alignment and ontology evolution are closely interrelated, because while ontology alignment is concerned with finding the

similarities in two ontologies, ontology evolution is concerned with finding the differences in two different versions of an ontology. These two tasks are complementary and it turns out that different functionalities from the area of ontology evolution can also be used for finding similarities across different ontologies [Noy and Musen, 2003]. Because of this close relationship, it would be useful to align the efforts in ontology alignment and ontology evolution in the Esperonto project in a next version.

Chapter 6

Conclusions and Outlook

We have introduced the problem of Information Integration and, more specifically, Information Integration at a Semantic level. Within enterprises, most information to be integrated is located in different disparate databases and other data sources. We have compared several classic, as well as novel, approaches to data modelling with ontologies and ontology modelling. It turns out that ontologies have a function in an open, environment, where databases operate in closed environments. Therefore, ontologies and (conceptual) database schemas have different characteristics.

Novel approaches to information integration, and, more specifically, semantic information integration, use ontology technology for the explication of the information in the enterprise and the annotation of the data sources (using explicit mappings to a central ontology) located throughout the enterprise.

In this context, we have introduced the COG project, where we faced the problems of Semantic Information Integration within an enterprise. We had to integrate disparate data sources, where the concepts in the data sources often had incomprehensible names, into a global unified Semantic Information Model (ontology). This global ontology, together with the mappings to the individual source schemas, enabled us to locate data in the organization, to query data located throughout the enterprise using a unified view and to translate instance data between disparate sources in the enterprise.

We have provided a comparison of the COG method and tool with other existing methodologies for Semantic Information Integration.

Taking the information integration one step further, we have looked into information integration between different organizations, or organizational units. The Semantic Web provides a platform for semantic information integration on an inter-organizational and even a global scale. We have introduced the Esperonto project, where the aim is to build an architecture, the SemASP (Semantic Annotation Service Provider), which enables annotation on the Semantic Web. In this project, we focussed on the inter-operation between ontologies on the Semantic Web, using a process of Ontology Alignment. We distinguished some problems in the area of ontology mapping and using these problems we have identified some requirements on such an Ontology Alignment solution.

The approaches in the COG project and in the Esperonto project have in common that they both use ontologies to model the domain of discourse and they both use Semantic Information Integration to enable inter-operation between

applications, and in the case of the Esperonto project, between organizations and organizational units.

6.1 Outlook

The development of a rule language (cf. RuleML, http://www.ruleml.org/) for the Semantic Web will enable specifying ontology mappings in a standardized way.

Database schemas currently in use will still need translation to a standardized (ontology) format in order to allow for mapping between database schemas and ontologies. Unfortunately, database schemas are often not so nicely made as they should have been. They often deviate from conceptual models and use obscure naming conventions. Furthermore, they lack formal and real-world semantics. As we have seen, semantic database schemas, as introduced by Ter Bekke [ter Bekke, 1992], do provide formal and real-world semantics. The database schema is specified here at a (high) conceptual level with a direct implementation; therefore, there is no loss of information that usually results from the translation from a conceptual schema to an internal schema. This provides for easier translation to an ontology language, because the concepts in the semantic data modelling language are close to the concepts generally used in ontology languages.

Nonetheless, the semantic database paradigm does not seem to be accepted by the database community and the major database vendors, such as IBM and Oracle. Some problems in relational databases are being solved by the SQL3 (discussed in [Elmasri and Navathe, 2000], section 13.4) in the class of the so-called Object-Relational Data Base Management Systems (ORDBMS). Major database management systems such as Oracle and PostgreSQL are embracing the Object-Relational model. It remains to be seen how database modelling will change with the rise of ORDBMS.

The view of the Semantic Web put forward in chapter 5 is still the view of a static Web, where agents combine knowledge in an intelligent way, but cannot initiate changes in this knowledge. Semantically annotated content can be retrieved and combined, but it is not possible to perform actions on the Semantic Web, that is, to initiate changes in the available knowledge.

Currently, there is a lot of interest in the research community in the area of Semantic Web Services [McIlraith et al., 2001]. Semantic Web Services add formal, explicit semantics to the description of Web Services, based on Semantic Web technologies, such as ontologies. An important aspect in the inter-operation of Semantic Web Services is data mediation (cf. [Fensel and Bussler, 2002]). As the data produced by Semantic Web Services in annotated with ontologies, there is an ontology mapping task at hand here.

Furthermore, during the invocation of Web Services, the data needs to be actually translated from one format to another. An explicit declarative mapping between ontologies can be used for such a purpose. The Unicorn Workbench, which we described in chapter 2, could be used for deriving such automatic translations. Currently, only database schemas can be mapped, but it might also be possible in the future to map arbitrary ontologies and provide translation rules for instances of the ontologies. The translation rules can be effectuated by a data mediation agent.

Bibliography

[Baader et al., 1991] Baader, F., Bürckert, H.-J., Heinsohn, J., Mller, J., Hollunder, B., Nebel, B., Nutt, W., and Profitlich, H.-J. (1991). Terminological knowledge representation: A proposal for a terminological logic. In *Proc. International Workshop on Terminological Logics, 1991.*

[Baader et al., 2003] Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., and Patel-Schneider, P. F., editors (2003). *The Description Logic Handbook.* Cambridge University Press.

[Bechhofer et al., 2003] Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F., and Stein, L. A. (2003). Owl web ontology language reference. Candidate Recommendation 18 August 2003, W3C.

[Bergamaschi et al., 2001] Bergamaschi, S., Castano, S., Beneventano, D., and Vincini, M. (2001). Semantic integration of heterogeneous information sources. *Special Issue on Intelligent Information Integration, Data & Knowledge Engineering*, 36(1):215–249.

[Bergamaschi et al., 1999] Bergamaschi, S., Castano, S., and Vincini, M. (1999). Semantic integration of semistructured and structured data sources. *SIGMOD Record Special Issua on Semantic Interoperability in Global Information*, 28(1).

[Berners-Lee et al., 2001] Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The semantic web. *Scientific American*, 284(5):34–43. http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21&ref=sciam.

[Borgida et al., 1989] Borgida, A., Brachman, R. J., McGuinness, D. L., and Resnick, L. A. (1989). Classic: A structural data model for objects. In *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, pages 59–67.

[Brachman and Schmolze, 1985] Brachman, R. and Schmolze, J. (1985). An overview of the kl-one knowledge representation system. *Cognitive Science*, 9(2):171–216.

[Brickley and Guha, 2003] Brickley, D. and Guha, R. V. (2003). Rdf vocabulary description language 1.0: Rdf schema. Working draft, W3C. http://www.w3.org/TR/2003/WD-rdf-schema-20030123/.

[Castano et al., 2001] Castano, S., Antonellis, V. D., and di Vimercati, S. D. C. (2001). Global viewing of heterogeneous data sources. *IEEE Transactions on Knowledge and Data Engineering*, 13(2):277–297.

[Chaudhri et al., 1998] Chaudhri, V. K., Farquhar, A., Fikes, R., Karp, P. D., and Rice, J. P. (1998). Okbc: A programmatic foundation for knowledge base interoperability. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pages 600–607, Madison, Wisconsin, USA. MIT Press.

[Chen, 1979] Chen, P. (1979). The entity relationship model - toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36.

[Date, 1984] Date, C. (1984). A critique of the sql database language. *ACM SIGMOD Record*, 14(3):8–52.

[de Boer and ter Bekke, 2001] de Boer, B. and ter Bekke, J. H. (2001). Applying semantic database principles in a relational environment. In Hamza, M. H., editor, *Proceedings of the IASTED International Symposia Applied Informatics (AI2001)*, pages 400–405, Innsbruck, Austria. IASTED/ACTA Press, Anaheim - Calgary - Zurich.

[de Bruijn, 2003] de Bruijn, J. (2003). Using ontologies. Study of literature, TU Delft. Available from http://homepage.uibk.ac.at/∼c703239/publications.html.

[de Bruijn et al., 2003a] de Bruijn, J., Ding, Y., and Arroyo, S. (2003a). Semantic information integration in the cog project. COG Project White Paper. To be published on http://www.cogproject.org/.

[de Bruijn et al., 2003b] de Bruijn, J., Ding, Y., Arroyo, S., and Lausen, H. (2003b). Ontology alignment solution. Internal Deliverable D1.4 v1.0, Esperonto Project IST-2001-34373.

[de Bruijn and Lausen, 2003] de Bruijn, J. and Lausen, H. (2003). Active ontologies for data source queries. COG Project White Paper. To be published on http://www.cogproject.org/.

[Ding and Foo, 2002] Ding, Y. and Foo, S. (2002). Ontology research and development, part 2 - a review of ontology mapping and evolving. *Journal of Information Science*, 28(5):375–388.

[Elmasri and Navathe, 2000] Elmasri, R. and Navathe, S. B. (2000). *Fundamentals of Database Systems, 3rd ed.* Addison Wesley.

[Fellbaum, 1999] Fellbaum, C., editor (1999). *WordNet: An Electronic Lexical Database*. MIT Press.

[Fensel, 2003] Fensel, D. (2003). *Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce, 2nd edition*. Springer-Verlag, Berlin.

[Fensel and Bussler, 2002] Fensel, D. and Bussler, C. (2002). The web service modeling framework wsmf. *Electronic Commerce Research and Applications*, 1(2):113–137.

[Fensel et al., 2001] Fensel, D., van Harmelen amd I. Horrocks amd D. L. McGuinness, F., and Patel-Schneider, P. (2001). Oil: An ontology infrastructure for the semantic web. *IEEE Intelligent Systems*, 16(2).

[Fernández-López, 1999] Fernández-López, M. (1999). Overview of methodologies for building ontologies. In *Workshop on Ontologies and Problem-Solving Methods: Lessons Learned and Future Trends (IJCAI99)*, pages 4–1 – 4–13.

[Fernández-López et al., 2002] Fernández-López, M., Gómez-Pérez, A., et al. (2002). A survey on methodologies for developing, maintaining, evaluating and reengineering ontologies. Deliverable 1.4, OntoWeb project (http://www.ontoweb.org/).

[Fowler et al., 1999] Fowler, J., Nodine, M., Perry, B., and Bargmeyer, B. (1999). Agent-based semantic interoperability in infosleuth. *SIGMOD Record*, 28(1).

[Gómez-Pérez et al., 2002] Gómez-Pérez, A. et al. (2002). A survey on ontology tools. Deliverable 1.3, OntoWeb project (http://www.ontoweb.org/).

[Gruber, 1993] Gruber, T. R. (1993). A translation approach to portable ontology specification. *Knowledge Acquisition*, 5(2):199–220.

[Hammer and McLeod, 1993] Hammer, J. and McLeod, D. (1993). An approach to resolving semantic heterogeneity in a federation of autonomous, heterogeneous, database systems. *International Journal on Intelligent and Cooperative Information Systems*, 2(1):51–83.

[Horrocks, 2002] Horrocks, I. (2002). DAML+OIL: a reason-able web ontology language. In *Proc. of EDBT 2002*, number 2287 in Lecture Notes in Computer Science, pages 2–13. Springer.

[Horrocks and Patel-Schneider, 2003] Horrocks, I. and Patel-Schneider, P. F. (2003). Reducing OWL entailment to description logic satisfiability. In *Proc. of the 2003 International Semantic Web Conference (ISWC 2003)*, Sanibel Island, Florida.

[Horrocks et al., 2003] Horrocks, I., Patel-Schneider, P. F., and van Harmelen, F. (2003). From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*. To appear.

[Horrocks and van Harmelen, 2001] Horrocks, I. and van Harmelen, F. (2001). Reference description of the daml+oil (march 2001) ontology markup language. Technical report, DAML. http://www.daml.org/2001/03/reference.html.

[Hull, 1997] Hull, R. (1997). Managing semantic heterogeneity in databases: A theoretical perspective. In *ACM Symposium on Principles of Database Systems*, pages 51–61, Tuscon, Arizona, USA.

[Kifer et al., 1995] Kifer, M., Lausen, G., and Wu, J. (1995). Logical foundations of object-oriented and frame-based languages. *JACM*, 42(4):741–843.

[Klein, 2001] Klein, M. (2001). Combining and relating ontologies: an analysis of problems and solutions. In Gomez-Perez, A., Gruninger, M., Stuckenschmidt, H., and Uschold, M., editors, *Workshop on Ontologies and Information Sharing, IJCAI'01*, Seattle, USA.

[Lara et al., 2003] Lara, R., Lausen, H., Arroyo, S., de Bruijn, J., and Fensel, D. (2003). Semantic web services: description requirements and current technologies. In *Semantic Web Services for Enterprise Application Integration and e-Commerce workshop (SWSEE03), in conjunction with ICEC 2003*, Pittsburgh, PA, USA.

[Lassila and Swick, 1999] Lassila, O. and Swick, R. R. (1999). Resource description framework (rdf) model and syntax specification. W3c recommendation, W3C. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222.

[McGuinness et al., 2000] McGuinness, D., Fikes, R., Rice, J., and Wilder, S. (2000). An environment for merging and testing large ontologies. In *Proc. 7th Intl. Conf. On Principles of Knowledge Representation and Reasoning (KR2000)*, Colorado, USA.

[McGuinness et al., 2003] McGuinness, D. L., Fikes, R., Stein, L. A., and Hendler, J. (2003). DAML-ONT: An ontology language for the semantic web. In Fensel, D., Hendler, J., Lieberman, H., and Wahlster, W., editors, *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential*, chapter 3, pages 65–93. MIT Press.

[McIlraith et al., 2001] McIlraith, S., Son, T., and Zeng, H. (2001). Semantic web services. *IEEE Intelligent Systems, Special Issue on the Semantic Web*, 16(2):46–53.

[Meersman, 2001] Meersman, R. (2001). Ontologies and databases: More than a fleeting resemblance. In *Rome OES/SEO Workshop*.

[Mena et al., 2000] Mena, E., Illarramendi, A., Kashyap, V., and Sheth, A. P. (2000). Observer: An approach for query processing in global information systems based on interoperation across pre-existing ontologies. *Distributed and Parallel Databases*, 8(2):223–271.

[Mitra and Wiederhold, 2001] Mitra, P. and Wiederhold, G. (2001). An algebra for semantic interoperability of information sources. In *IEEE International Conference on Bioinformatics and Biomedical Egineering*, pages 174–182.

[Mitra et al., 1999] Mitra, P., Wiederhold, G., and Jannink, J. (1999). Semi-automatic integration of knowledge sources. In *Proceedings of Fusion 99*, Sunnydale, California, USA.

[Mitra et al., 2000] Mitra, P., Wiederhold, G., and Kersten, M. (2000). A graph-oriented model for articulation of ontology interdependencies. In *Proceedings of Conference on Extending Database Technology (EDBT 2000)*, Konstanz, Germany.

[Nodine et al., 2000] Nodine, M. H., Fowler, J., Ksiezyk, T., Perry, B., Taylor, M., and Unruh, A. (2000). Active information gathering in infosleuth. *International Journal of Cooperative Information Systems*, 9(1-2):3–28.

[Noy and Musen, 1999] Noy, N. F. and Musen, M. A. (1999). Smart: Automated support for ontology merging and alignment. Technical Report SMI-1999-0813, Stanford Medical Informatics.

[Noy and Musen, 2000a] Noy, N. F. and Musen, M. A. (2000a). Anchor-prompt: Using non-local context for semantic matching. In *Proceedings of the Workshop on Ontologies and Information Sharing at the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-2001)*, Seattle, WA, USA.

[Noy and Musen, 2000b] Noy, N. F. and Musen, M. A. (2000b). Prompt: Algorithm and tool for automated ontology merging and alignment. In *Proc. 17th Natl. Conf. On Artificial Intelligence (AAAI2000)*, Austin, Texas, USA.

[Noy and Musen, 2003] Noy, N. F. and Musen, M. A. (2003). Ontology versioning as an element of an ontology-management framework. To be published in IEEE Intelligent Systems.

[Patel-Schneider et al., 2003] Patel-Schneider, P. F., Hayes, P., and Horrocks, I. (2003). Owl web ontology language semantics and abstract syntax. Candidate Recommendation 18 August 2003, W3C.

[Preece et al., 2001] Preece, A. D., Hui, K.-Y., Gray, W. A., Marti, P., Bench-Capon, T. J. M., Cui, Z., and Jones, D. (2001). Kraft: An agent architecture for knowledge fusion. *International Journal of Cooperative Information Systems*, 10(1-2):171–195.

[Rahm and Bernstein, 2001] Rahm, E. and Bernstein, P. A. (2001). A survey of approaches to automatic schema matching. *VLDB Journal: Very Large Data Bases*, 10(4):334–350.

[Schreiber, 2003] Schreiber, Z. (2003). Semantic information management: Solving the enterprise data problem. To be found on the http://www.unicorn.com/ website.

[Studer et al., 1998] Studer, R., Benjamins, V. R., and Fensel, D. (1998). Knowledge engineering: Principles and methods. *Data and Knowledge engineering (DKE)*, 25(1-2):161–197.

[Stumme and Maedche, 2001] Stumme, G. and Maedche, A. (2001). Fca-merge: Bottom-up merging of ontologies. In *7th Intl. Conf. on Artificial Intelligence (IJCAI '01)*, pages 225–230, Seattle, WA, USA.

[ter Bekke, 1992] ter Bekke, J. H. (1992). *Semantic data modeling*. Prentice Hall, Hemel Hempstead.

[ter Bekke, 1997a] ter Bekke, J. H. (1997a). Can we rely on sql? In Wagner, R., editor, *Proceedings 8th international DEXA workshop*, pages 378–383, Toulouse. IEEE Computer Society.

[ter Bekke, 1997b] ter Bekke, J. H. (1997b). Comparative study of four data modeling approaches. In Siau, K., Wand, Y., and Parsons, J., editors, *Proceedings 2nd international EMMSAD workshop*, pages B1–B12, Barcelona.

[Tsichritzis and Klug, 1978] Tsichritzis, D. and Klug, A., editors (1978). *The ANSI/X3/SPARC DBMS Framework*. AFIPS Press.

[Ullman, 1988] Ullman, J. D. (1988). *Principles of Database and Knowledge-Base Systems, Volume I*. Computer Science Press.

[Unicorn, 2003a] Unicorn (2003a). *Unicorn v2.5 Modeling Guide*. Unicorn.

[Unicorn, 2003b] Unicorn (2003b). *Unicorn™ Workbench v2.5 User Manual*. Unicorn.

[Uschold, 2000] Uschold, M. (2000). Creating, integration, and maintaining local and global ontologies. In *Proceedings of the First Workshop on Ontology Learning (OL-2000) in conjunction with the 14th European Conference on Artificial Intelligence (ECAI-2000)*, Berlin, Germany.

[Visser and Cui, 1998] Visser, P. and Cui, Z. (1998). On accepting heterogeneous ontologies in distributed architectures. In *Proceedings of the ECAI98 workshop on applications of ontologies and problem-solving methods*, Brighton, UK.

[Visser and Tamma, 1999] Visser, P. and Tamma, V. (1999). An experience with ontology clustering for information integration. In *Proceedings of the IJCAI-99 Workshop on Intelligent Information Integration in conjunction with the Sixteenth International Joint Conference on Artificial Intelligence*, Stockholm, Sweden.

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

[Wiederhold, 1992] Wiederhold, G. (1992). Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):38–49.

[Wiederhold, 1994] Wiederhold, G. (1994). An algebra for ontology composition. In *Proceedings of 1994 Monterey Workshop on formal Methods*, pages 56–61, U.S. Naval Postgraduate School, Monterey CA.

[Wiederhold and Genesereth, 1997] Wiederhold, G. and Genesereth, M. R. (1997). The conceptual basis for mediation services. *IEEE Expert*, 12(5):38–47.