UNIVERSITÀ DEGLI STUDI DI MODENA
E REGGIO EMILIA
Facoltà di Ingegneria - Sede di Modena
Corso di Laurea in Ingegneria Informatica

Ontology dynamics for Semantic Web: the MOMIS approach

Dinamica delle ontologie per il Web Semantico: l'approccio di MOMIS

Relatore
Chiar.mo Prof. Sonia Bergamaschi

Tesi di Laurea di
Alain Fergnani

Correlatore
Dott. Ing. Francesco Guerra

Controrelatore
Chiar.mo Prof. Paolo Tiberio

Anno Accademico 2001 - 2002

Key words:
Semantic Web
Ontology
Dynamics
MOMIS
SEWASIE

RINGRAZIAMENTI

Desidero ringraziare la Professoressa Sonia Bergamaschi, l'Ing. Francesco Guerra, l'Ing. Maurizio Vincini e il Professore Domenico Beneventano per l'aiuto fornito durante la realizzazione della presente tesi.
Ringrazio inoltre l'A.C. che mi ha permesso di avere sempre una buona scusa per staccare dallo studio.
Un ringraziamento speciale va poi alla mia famiglia che mi ha permesso di raggiungere questo importante obiettivo e alla mia fidanzata, Letizia, che mi ha sempre sostenuto e sopportato in questi anni di studi.

Contents

List of Figures

List of Tables

Abstract

Nowadays the Web is a huge collection of information and its expansion rate is very high. The users need new ways to exploit all this available information and possibilities. The problem is that Web information is meaningless for a computer and so it is very hard to find out what we are looking for.
The Semantic Web is the new challenge of web using. Its goal is to bring structure and semantics to the content of Web pages, creating an environment where software agents, roaming from page to page, can carry out sophisticated tasks for users.
In consequence of this new vision of the Web many research groups began to study related problems. One of the most important seems to be the *Ontologies management*. In fact, Ontologies are considered as a key component to build the Semantic Web.
The present work explains the new scenario of the Semantic Web by means of the numerous projects involved (mainly W3C). Moreover, we present some projects related to the ontology issue and, in particular, *the ontologies dynamics problem*.
In this scenario we present **SEWASIE** (**SE**mantic **W**ebs and **A**gent**S** in **I**ntegrated **E**conomies), a new project funded by the UE and led by the Modena and Reggio Emilia University. Finally we explain the relevance of the **MOMIS** (**M**ediator envir**O**nment for **M**ultiple **I**nformation **S**ources) system, an Intelligent Information Integration system developed at the University of Modena and Reggio Emilia, and in particular the attempts to solve the problems of managing dynamic aspects in an ontology.

Introduction

The current WWW is an enormous amount of documents and information and its growing rate is very high. Besides, more and more users and businesses have access to the Web and want to find what they need in a short time. In the last few years the rapid increase in the available information has complicated the search of documents and web pages semantically rich for users. In fact, if on one hand internet search engines are fundamental for retrieving information, on the other hand the search results are often meaningless and completely out of the domain users are interested in. Moreover from the business point of view, it is more and more difficult to keep their data updated and to make them available for clients in the right format.

In this context arises the need for a *new vision of the Web*; its inventor Tim Berners-Lee has called it **The Semantic Web** [1]. Berners-Lee, that has also built the infrastructure of the current web, has imagined a Web where resources could be annotated with machine-processable metadata providing them with background knowledge and meaning. Moreover this annotation could be exploited by users' agents, that can carry out sophisticated tasks and searches roaming from page to page.

This new scenario creates many expectations among the users and information providers but new issues and new problems have to be solved before achieving good results. Thus universities, private research groups, enterprises began to study possible solutions. One of the main issue in this context is "**dynamics**". Web environment is very changeable, it is continuously updated, modified and the users need to rely on the data they retrieve from the net. Another fundamental component seems to be **ontology**; this "*explicit specification of a conceptualization*" [2] lets information providers annotating their domains. *The annotation phase* is a crucial step in order to create a semantically rich environment exploitable and intelligible by users' agents. Many studies are trying to find *languages* and *standards* that can help domain experts in the delicate task of expressing their knowledge in a formal

way. But as in the case of data, ontologies evolve, and therefore we have to face again the problem of **managing the dynamics** with respect to ontologies.

Another important aspect to be considered in this context is: *sources integration*. Today the ability to integrate, merge, manipulate heterogeneous sources is fundamental; companies merge their departments, doctors gather their data, vendors want to create common market places, etc.

It is clear by now that the problem is really complicated and it results in a lot of related implications, such as: information retrieval and integration, semantic enrichment and dynamics management. For these reasons, the main purpose of this work is: first, to better understand how international research projects face the mentioned problems; second to study the state of the art of the tools for ontologies management, in order to acquire important knowledge for the solution of our problem, and finally to elaborate a solution to effectively manage the insertion of new sources in the context of the MOMIS system, which is the intelligent information integration system developed at the University of Modena and Reggio Emilia.

Main organization of this dissertation.

Chapter 1
This chapter introduces the *Semantic Web* with a typical scenario of the new WWW.

Chapter 2
The SEWASIE (SEmantic Webs and AgentS in Integrated Economies) project is presented; its main objectives, features and architecture are introduced. The second part of the chapter is dedicated to explain the MOMIS (Mediator envirOnment for Multiple Information Sources) system; in particular the semantic MOMIS approach to sources integration and the developed architecture of the framework are explained.

Chapter 3
This chapter describes the important role of "*ontologies*" in Semantic Web. Many definitions and classifications are presented. The fundamental problem of the *ontology dynamics* is taken into consideration describing two possible approach: *evolution* and *versioning*.

Chapter 4

We dedicate this chapter to explore the main *tools and projects for ontology management*. We define the principal requirements for ontologies' editors, and we review the principal tools for developing and integrating ontologies. Moreover, we present *SI-Designer* (the GUI of the MOMIS system for the integration process) that is the MOMIS tool for building and manage an ontology.

Chapter 5
This chapter, starting from the MOMIS approach, presents the core contribution of this dissertation, i.e. the extension of the MOMIS approach for the management of the dynamics of an ontology.

Chapter 6
The last chapter describes some of the most important Semantic Web projects. We present also some new languages (OWL and SHOE) for the management of ontologies.

Finally we give some remarks in chapter *Conclusions and Future Works*, a complete running example in *Appendix A* and the principal features of ODL_{I^3} language in *Appendix B*. *Appendix C* gathers the links present within this thesis.

Chapter 1

The Semantic Web

In this chapter we will present the new scenario introduced by a new vision of the web: **the Semantic Web**. A fundamental paper about the Semantic Web is the one written by J. Hendler, O. Lassila and T. Berners-Lee that appeared on Scientific America in 2001 [1]. The authors depicted a new "world" where people interoperate with their electronic devices in a completely different way from today. They imagine that, for instances, you could be at the doctor's office and you have to book a series of therapy sessions but you already have some appointments. You can instruct your Semantic Web agent through your handheld Web browser. You can set up the search to find only places within a 20-km radius of your home and only in a specific time of the day. The agent, then, starts to look for information interacting with other agents present on the web sites of the doctors in the area. In few minutes the agent presents you with a plan; if you like it you can change your week agenda and mail your tennis trainer that for a month you can not play. If you don't like the plan you can set up your agent with stricter preferences and redo the search.

At present the World Wide Web cannot carry out all the tasks of the above scenario. Most of the Web's content today is designed for humans interaction, not for computer programs. Computers can easily parse Web pages for layout and routine processing – a header, a link to another page – but in general they have no reliable way to process the semantics: this is the home page of my professor and this link goes to the course material page.

This is the reason for which the research area of Semantic Web is active both at the the European (IST program supports several projects, On-To-Knowledge, SEWASIE, etc.) and American level as it is shown in the following subsections.

1.1 What is the Semantic Web?

Nowadays the Web is a huge collection of information useful for people but meaningless for machines without a post-processing. By now the WWW reaches its second version; the first saw the HTML static pages changing the style people share information, while the second is the one we are experiencing today with dynamic pages created as response to a query on a database. The increasing of available information and of the usage of the Web, are creating new expectations with the users. In other words it's time to think about the third version of the WWW: ***the Semantic Web***.
The Semantic Web will bring structure to the meaningful content of Web pages, creating an environment where software agents, roaming from page to page, can readily carry out sophisticated tasks for users.
Such an agent coming to the professor's Web page will not just know that the page has some keywords such as "ontology, knowledge representation, artificial intelligence" (as might be encoded today) but also that the course is scheduled for Tuesday and Thursday at 11.00 am. These semantics were encoded into the Web pages using off-the-shelf software for writing Semantic Web pages.
The Semantic Web is not a separate Web but an extension of the current one, in which information is given well-defined meaning, better enabling computers and people to work in cooperation. The first steps in weaving the Semantic Web into the structure of the existing Web are already under way. In the near future, these developments will introduce new functionality as machines become much better able to process and "understand" the data that they merely display at present.
The essential property of the World Wide Web is its universality. The power of a hypertext link is that "anything can link to anything". Web technology, therefore, does not discriminate between commercial and academic information, or among cultures, languages, media and so on. Information varies along many axes. One of these is the difference between information produced primarily for human consumption and that produced mainly for machines. At one end we have everything from the TV commercial to poetry. At the other end we have databases, programs and sensor output. Today, the Web has developed most rapidly as a means of documents for people rather than for data and information that can be processed automatically. The Semantic Web aims to build these capabilities. Like the Internet, the Semantic Web will be as decentralized as possible. Such Web-like systems generate a lot of excitement at every level, from major corporation to individual user, and provide benefits that are hard or impossible to predict in advance.

1.2 Knowledge representation

For the semantic web to function, computers have to be able to access structured collections of information and sets of inference rules that they can use to conduct automated reasoning. Artificial-intelligence and DataBase researchers have studied such systems since long before the Web was developed. Knowledge representation is currently in a state comparable to that of hypertext before the advent of the Web. Many studies has been conducted in this area but till now there are only some demonstrations of the numerous potential.
Traditional knowledge-representation systems typically have been centralized, requiring everyone to share exactly the same definition of common concepts. But central control is increasing the size and scope of such a system and rapidly it becomes unmanageable.
Semantic Web researchers make the language for the rules as expressive as needed to allow the Web to reason as widely as desired. The challenge of the Semantic Web, therefore, is to provide a language that expresses both data and rules for reasoning about the data and that allows rules from any existing knowledge-representation system to be exported onto the Web.
Adding logic to the Web – that means to use rules to make inferences, to choose courses of action and to answer question – is a fundamental task within the Semantic Web community at the moment. Two important technologies for developing the Semantic Web are already in place: **eXtensible Markup Language** (**XML**) and the **Resource Description Framework** (**RDF**). XML lets everyone create their own tags – hidden labels that annotate Web pages or sections of text on a page. Scripts, or programs, can make use of these tags in sophisticated ways, but the scripts writer has to know what the page writer uses each tag for. In other words, XML allows users to add arbitrary structure to their documents but says nothing about what the structures mean.
Meaning is expressed by RDF, which encodes it in *sets of triples*, each triple being like subject, verb and object of an elementary sentence. These triples can be written using XML tags. In RDF, a document makes assertions that is, particular things (people, Web pages or whatever) have properties ("is a sister of", "is the author of") with certain values (another person, another Web page). This structure turns out to be a natural way to describe the vast majority of the data processed by machines. Subject and object are each identified by a Universal Resource Identifier (URI), just as used in a link on a Web page. The verbs are also identified by URIs, which enables anyone to define new concept, a new verb, just by defining a URI for it somewhere on the Web.

1.3 Ontologies

In such a system, where information is distributed in different Web sites or databases, a program that wants to compare or combine information across two databases has to solve the problem of information integration. For example the word "staff" in a DB may reference to the people which work in a certain department while another DB may call them "employee". Ideally, the program must have a way to discover such common meanings for whatever databases it encounters. A solution to this problem is provided by the key component of the Semantic Web, collections of information called *ontologies*. ***Ontology*** is a term borrowed from philosophy that refers to the science of describing the kinds of entities in the world and how they are related (see chapter 3 for a complete overview). The most typical kind of ontology for the Web has a taxonomy and a set of inference rules. The taxonomy defines classes of objects and relations among them. For example, an address may be defined as a type of location, and city codes may be defined to apply only to locations, and so on. Classes, subclasses and relations among entities are a very powerful tool for Web use. We can express a large number of relations among entities by assigning properties to classes and allowing subclasses to inherit such properties.

Inference rules in ontologies supply further power. An ontology may express rules on the classes and relations in such a way that a machine can deduce some conclusions. The computer does not truly "understand" any of this information, but it can now manipulate the terms much more effectively in ways that are useful and meaningful to the human user. With ontology pages on the Web, solutions to terminology (and other) problems begin to emerge. The meaning of terms or XML codes used on a Web page can be defined by pointers from the page to an ontology.

Ontologies can enhance the functioning of the Web in many ways. They can be used in a simple fashion to improve the accuracy of Web searches – the search program can look for only those pages that refer to a precise concept instead of all the ones using ambiguous keywords. More advanced applications will use ontologies to relate the information on a page to the associated knowledge structures and inference rules.

A more precise and large description of ontologies will be presented in chapter two.

1.4 Agents

The real power of the Semantic Web will be realized when people create many programs that collect Web content from diverse sources, process the information and exchange the results with other programs. The effectiveness of such software agents will increase exponentially as more machine-readable Web content and automated services (including other agents) become available. In fact, an agent is a software component that enjoys the following properties [3]:

- autonomy: agents encapsulate some state and make decision about what to do based on this state, without any human interaction;
- reactivity: agents are situated in an environment, they are able to perceive the modifications of the environment, and they can respond to those modifications;
- pro-activeness: agents are able to exhibit goal-directed behavior by taking the initiative;
- social ability: agents interact with other agents and with other applications.

Due to the described features, an agent can automatically navigate in the Web and move from a site to an other site in order to accomplish a specific task. If the web where the agent navigates is a Semantic Web, an agent may have the perception of the contents stored in the visited sources. In this way, it can change its behavior on the base of the visited sites, and, for example, it can follow specific links because they are semantically correct with respect to the goal of the agent. Therefore, the Semantic Web allows an agent to have a well-knowledge of the environment where the agent is instantiate and consequently to react to the environment without mistakes.

Important contributions to these fundamental components of the Semantic Web arrive from **AgentLink**[1], an European excellence network funded by the UE IST programme, that is very active in the study and productions of software agents.

[1] http://www.agentlink.org/

Chapter 2

SEWASIE and MOMIS

2.1 SEWASIE

2.1.1 Overview

SEWASIE[1] stands for "**SEmantic Webs and AgentS in Integrated Economies**". It is an European project inserted in the 5^{th} Framework IST programme of the European Community within the Semantic Web Action Line. The project is coordinated by Università degli studi di Modena e Reggio Emilia (Italy), and the other partners are: CNA SERVIZI Modena s.c.a.r.l. (Italy), Università di Roma "La Sapienza" (Italy), Rheinisch Westfaelische Technische Hochschule Aachen (Germany), The Victoria University of Manchester (UK), Thinking Networks AG (Germany), IBM Italia SPA (Italy), Fraunhofer-Gesellschaft zur Frderung der angewandten Forschung eingetragener Verein (Germany). It started in May 2002 and will end in April 2005.

SEWASIE *will design and implement an advanced search engine enabling intelligent access to heterogeneous data sources on the web via semantic enrichment to provide the basis of structured secure web-based communication.*

The objectives are to develop a distributed agent-based architecture of semantic search and communication using community-specific multilingual ontologies; to equip ontologies with an inference layer grounded in W3C standards; to develop prototypes that meet the needs of SMEs in a EU context; to obtain practical experience (user requirements, potential added value, risks

[1] http://www.sewasie.org/

etc.). In particular the programme focuses on the question how to assist networks of small and medium enterprises (so-called Integrated Economies) in enhancing their intra-and inter-organisational information management capabilities. The project also includes novel techniques for semantic enrichment, query management, and presentation techniques in multi-lingual information acquisition from the web.

2.1.2 The Business Scenario

Throughout Europe, much of the industrial fabric is made of small and medium-sized agricultural, manufacturing, commercial, and services firms (SMEs). For social and historical reasons, they often aggregate into sectorial clusters (that the economic literature names as industrial districts); nowadays, this kind of economic organization is made vulnerable by the globalisation and it will be more so in the future.

In such a condition, to find (the adequate supplier, an innovative working method, a new market, and so on) and to be found (by possible customers, partners or sponsors) makes the difference; the current Internet tools (e.g. search engines) are inadequate since they are very difficult to use (small firms do not necessarily have the required technological infrastructure and know-how) and because a simple query produces pages and pages of links, most of them worthless. Suppose that a firm needs to know about a topic (a product, a supplier, a fashion trend, a norm, and so on): generally they have to make several quests. For example, if they look for a new "fabric dyeing process" they need to find who produces it and if there exist specific norms about the disposal of the dyeing waste material. First of all they have to make a quest using the term "fabric dyeing". Clearly, the search engine will list links (540 in the case of www.google.com) concerning not only manufacturers of fabric dyeing equipment, but also the history of dyeing, the dyeing technology, and so on. After having located a possible supplier, the company must find relevant laws and norms about waste disposal and their related interpretations. The search criteria can be, for example, the number of the law, the term "law", the interpretation etc. Both searches become more difficult when searching abroad, since there can exist a specific terminology that the user might not be familiar with.

Suppose instead that the user is a business professional, and (s)he is interested only in the normative part. In this case (s)he needs to retrieve rapidly not only the relevant laws, but also references, connected issues and so on; all this by searching both on free and subscribed web sites.

For this reason, users need to have at their disposal an engine equipped with an easy-to-use query interface able to extract the required informa-

tion from the Internet and to show it in an easily enjoyable format. If a step-by-step approach is allowed, the following deductive reasoning steps are easy as well. Easy-to-use means "usable by lay users and by means of poor infrastructures".

The main requirement is to get structured results obtained from an interpretation of vague queries followed by some filtering techniques based on designer rules and the acquired experiences. For instance, starting from a request simply constituted by the keyword "punch" the engine should answer with one or more documents containing information in an easy-to-accessible format such as sellers, prices, manufacturers, technical literature on its use, importer and so on. Particularly interesting for products and manufactures is also to know if some stock goods at low price exist or if there are any auctions or other negotiation mechanisms for the construction or the purchase (from e-procurement sites).

2.1.3 The SEWASIE architecture

SEWASIE tools and methods will be developed to create/maintain multilingual ontologies, with an inference layer grounded in W3C standards (XML, XML Schema, RDF(S)), that are the basis for the advanced search mechanisms and that provide the terminology for the structured communication exchanges. Search results will be personalised and visualised according to users' preferences.

SEWASIE will provide an open and distributed architecture based on mobile, intelligent agents (brokers, mediators and wrappers) facing scalability and flexibility issues, i.e. the ability to fit in changing and growing environments and to interoperate with other systems, while offering one central point of access to the user.

The user will be supported in querying heterogeneous web information sources. The user query will be sent to a *query agent* that processes and answers the query moving through the SEWASIE information nodes in order to retrieve the information requested by the user (see figure 2.1).

Information nodes are independent components that semantically enrich existing data sources by linking the data to ontologies and other metadata. The SEWASIE project will also allow for a real life business evaluation of the results, striving to develop a system and tools which will not only solve the problem, but do so in a usable, marketable way.

In particular, the SEWASIE project will pursue the following aims:

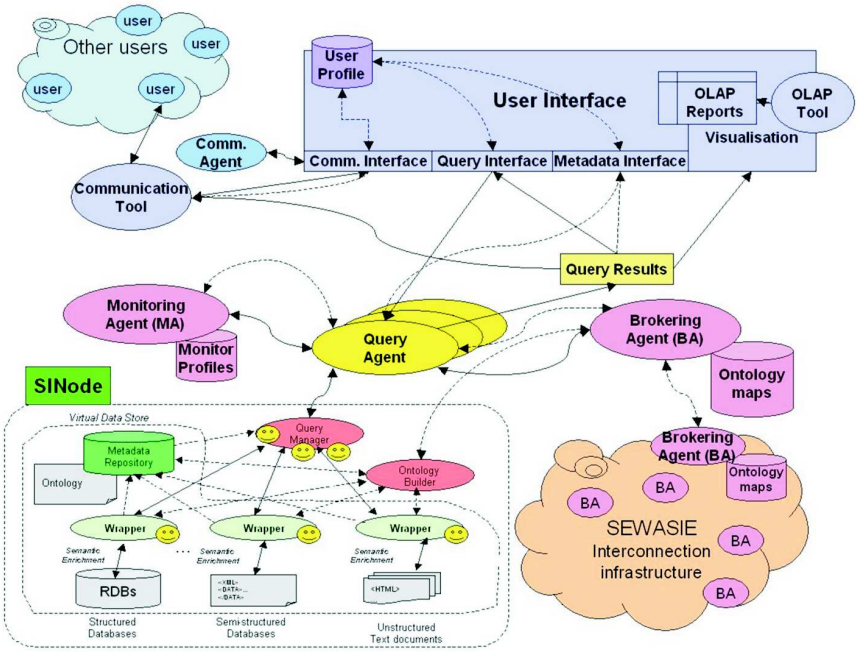

Figure 2.1: The SEWASIE Virtual Network

- To develop an agent-based secure, scalable and distributed system architecture for semantic search (ontology based) and for structured web-based communication (for electronic negotiation).

- To develop a general framework responsible for the implementation of the semantic enrichment processes leading to the semantically enriched virtual data stores, which constitute the information nodes accessible by the users. The created ontology will be multilingual accessible, founded on a logical layer and coded using widespread W3C standards.

- To develop a general framework for query management and information reconciliation taking into account the semantically enriched data stores. In particular, first commonalities among queries will be detected, then the relevant virtual data stores responsible for answering parts of the queries determined and the queries accordingly splitted, finally the sub-answers will be combined together in order to provide the user with an overall answer to the original query.

- To develop an information brokering component which will include methods for collecting, contextualising and visualising semantically rich

data. To obtain this result, intelligent information filtering and knowledge guidance services will be developed on the basis of semantic web technologies; structured data will be linked to semi- or unstructured data via ontologies; for financial controlling applications collected data will be visualised to show related documents and search result contexts.

- To develop structured communication processes that enable the use of ontologies. The communication tool will enable structured negotiation support for human negotiators engaging in business-to-business electronic commerce and employing intelligent software agents for some routine communication task.

- To develop end-user interfaces for both the semantic design and the query management. The first is a tool supporting the design, the management, and the storage of the semantic information associated to virtual data stores together with a conceptual modelling methodology associated to the devised data model. The latter is a tool for end-user query management and intelligent navigation exploiting the semantic information associated to virtual data stores and to the global virtual view.

Another important aim of the project is to assess the economic and organisational impact of a semantically rich web of information on SME industrial systems. In particular the potential economic benefits expected by the SEWASIE system and the internal and external factors needed to achieve such benefits will be analyzed, taking care to study the business and organisational changes requested to the firms by the new tool. Since different firms exhibit different features concerning the required type of information, available information sources, own skills, manufacturing technologies and so on, SEWASIE has selected to conduct the business user requirement analysis and test in two very distinct sectors, namely the industrial moulding sector and the textile and clothing.

2.2 MOMIS

2.2.1 Overview

In the last years the need to have access to distributed information and the problem of data integration coming from heterogeneous sources have become more and more important. Companies have equipped themselves with data storing systems building up informative systems containing data which are

related one another, but which are often redundant, heterogeneous and not always substantial. On the other hand, the web explosion, both at internet and intranet level, has enlarged the need for the sharing and retrieving of information located in different sources, thus obtaining an integrated view so as to eliminate any contradiction or redundancy. The problems that have to be faced in this field are mainly due both to structural and application heterogeneity, as well as to the lack of a common ontology, causing semantic differences between information sources. Moreover these semantic differences can cause different kinds of conflicts, ranging from simple contradictions in names' use (when different names are used by different sources to indicate the same concept), to structural conflicts (when different models/primitives are used to represent the same information). The integration problem is relevant also in the E-Commerce environment. Electronic catalogs are a key component of E-Commerce and they can be organized as individual company catalogs or they can participate in a multicatalog framework. In the second case, from a user point of view, it is very important to have a uniform interface to search products, that is a uniform view of data coming from different companies catalogs and a unique query language. On the other hand, from a company point of view it is important to guarantee both the uniqueness of their catalogs and the participation in a multi-catalog framework. Virtual Catalogs synthesize this approach as they are conceived as instruments to dynamically retrieve information from multiple catalogs and present product data in a unified manner, without directly storing product data from catalogs. Instead of having to interact with multiple heterogeneous catalogs, customers can interact in a uniform way with a virtual catalog.

In this context the problem to be faceed is the identification of semantically related information, that is, information describing the same real-world concepts in different sources having semantic heterogeneity. In fact, information sources to be integrated are usually preexisting and have been developed independently. Consequently, semantic heterogeneity can arise for the aspects related to terminology, structure, and context of the information, and has to be properly dealt with during integration in order to effectively and correctly exploit the information available at the sources.

MOMIS[2] stands for "**M**ediator envir**O**nment for **M**ultiple **I**nformation **S**ources". MOMIS [4, 5, 6, 7] is a mediator-based system for information extraction and integration that works with structured and semistructured data sources. MOMIS began as a joint collaboration among the University of Modena and Reggio Emilia, the University of Milano and the University of Brescia, within the INTERDATA national research project, under the

[2]http://www.dbgroup.unimo.it/Momis/

direction of Professor S. Bergamaschi. Now, part of the research activity continues within the *D2I: Integration, Warehousing, and Mining of Heterogeneous Data Sources* national research project.

2.2.2 The MOMIS approach

MOMIS [5, 6, 7] follows a "semantic approach" to information integration based on the conceptual schema (or metadata), of the information sources. In the MOMIS system, each data source provides a schema and a global virtual schema of all the sources is semi-automatically obtained. The global schema has a set of mapping descriptions that specify the semantic mapping between the global schema and the sources schemata.

The system architecture is composed of functional elements that communicate using the CORBA standard. A data model, ODM_{I^3}, and a language, $\text{ODL}_{I^3}$are used to describe information sources. $\text{ODL}_{I^3}$and ODM_{I^3}have been defined as a subset of the corresponding ones in ODMG, augmented by primitives to perform integration. To interact with a specific local source, MOMIS uses a Wrapper, which has to be placed over each source. The wrapper translates metadata descriptions of a source into the common ODL_{I^3}representation. The Global Schema Builder (GSB) module (see Figure 2.2) processes and integrates descriptions received from wrappers to derive the global shared schema by interacting with different service modules: **ODB-Tools**, an integrated environment for reasoning on object oriented database that relies on Description Logics [8]; **WordNet** [9] a lexical database that supports the mediator in building lexicon-derived relationships; and **ARTEMIS** tool that performs the clustering operation [10].

2.2.3 The architecture

The MOMIS system is designed for schema integration. Figure 2.2 shows the architecture of the system in terms of functional modules. The system is composed of several objects that communicate using the CORBA[3] standard and it follows the I^3architecture [11]. The modules are on three levels:

Data Level. Here there are the **Wrappers**. They are placed over each source and represent the interface modules between the mediator and the local data sources. They carry out a double function: in the integration phase, they translate the description of the information held in the source. This description is supplied through the ODL_{I^3} language; in the query processing phase, they translate the query which has been

[3]Object Management Group. http://www.omg.org/

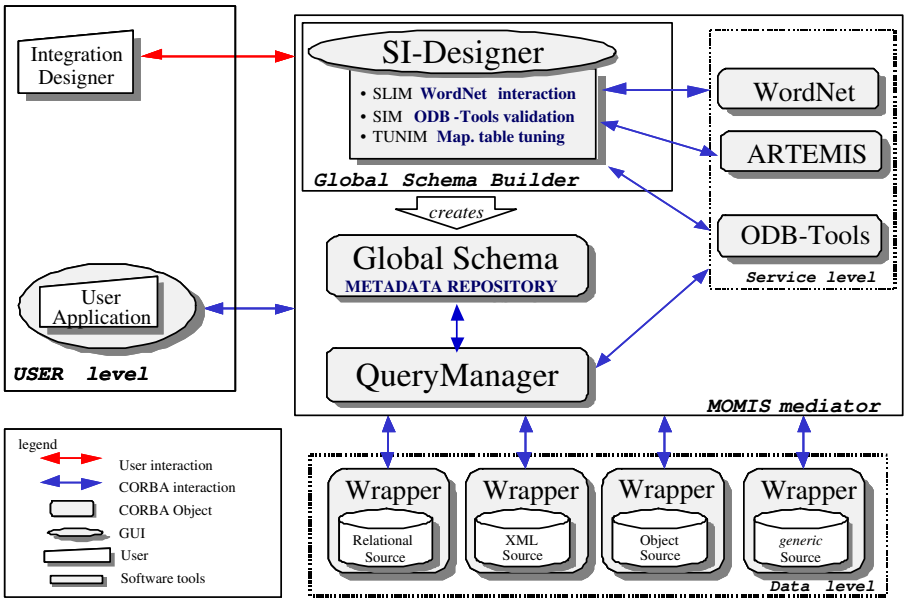

Figure 2.2: MOMIS prototype architecture

received by the mediator (expressed in the common query OQL_{I^3} language, derived by the OQL language) in a query expressed in the source query language. The wrappers must also export query result data, providing them to the mediator through the data common model used by the system.

Mediator Level. The mediator is the core module of the system. It allows to create an homogeneous representation of the information and an integrated access to the sources. It's composed of two distinct modules:

- *The Global Schema Builder* (GSB): its goal is to lead the designer from the schemata acquisition of sources to the tuning of the mapping table through the various steps of the integration. This module interacts with ODB-Tools (a framework for object-oriented database schema validation, preserving taxonomy coherence and performing taxonomic inference, and semantic query optimization) and with the lexical database WordNet to extract lexicon derived intensional relationships between attributes and classes. **SI-Designer** is the GUI (Graphic User Interface) for the Global Schema Builder.
- *The Query Manager* (QM): it is the module that performs query

processing and optimization. The QM generates OQL_{I^3} queries to be sent to wrappers starting from each query posed by the user on the Global Schema. It automatically generates the translation of the query into a corresponding set of sub-queries for the sources and synthesizes a unified global answer for the user.

User Level. The designer interacts with the Global Schema Builder and builds the integrated view of the sources. The users pose the queries on the Global Schema then the QM performs the queries on the local sources and presents the users with an unified answer.

The MOMIS project aims at integrating data from structured and semi-structured data sources. SI-Designer is a support tool for semi-automatic integration of heterogeneous sources schema (relational, object, XML and semistructured sources).

The MOMIS approach to perform Global Virtual View is articulated in the following phases:

1. *Generation of a Common Thesaurus.*
   The Common Thesaurus is a set of terminological intensional and extensional relationships, describing intra and inter-schema knowledge about classes and attributes of sources schemas. We express intra and inter-schema knowledge in form of terminological and extensional relationships (*synonymy, hypernymy and relationship*) between classes and/or attribute names. In this phase the WordNet database is used to extract lexicon derived relationships.

2. *Affinity analysis of classes.*
   Relationships in the *Common Thesaurus* are used to evaluate the level of *affinity* between classes intra and inter sources. The concept of affinity is introduced to formalize the kind of relationships that can occur between classes from the integration point of view. The affinity of two classes is established by means of affinity coefficients based on class names, class structures and relationships in the *Common Thesaurus*.

3. *Clustering classes.*
   Classes with affinity in different sources are grouped together in clusters using hierarchical clustering techniques. The goal is to identify the classes that have to be integrated because describing the same or semantically related information.

4. *Generation of the mediated schema.*
   Unification of affinity clusters leads to the construction of the predicted

schema. A class is defined for each cluster, which is representative of all clusters' classes and is characterized by the union of their attributes. The global schema for the analyzed sources is composed of all classes derived from clusters, and is the basis for posing queries against the sources.

A graphical tool, the Source Integration Designer, SI-Designer, has been developed to support the designer during the source integration phase. More details about SI-Designer will be presented in section 4.1.

Chapter 3

Ontologies

Many articles in the literature have been devoted to presenting different definitions and features of ontologies; in this chapter we have followed the overview written by Valentina Tamma in her Ph.D. Final Thesis [12].

The need to share knowledge and information with other applications already built, has given rise to a growing interest in research on ontology. This term has been originally used in Philosophy where it indicated the systematic explanation of Existence. More recently, the term has been used in various areas in Artificial Intelligence (AI) and more widely in Computer Science, such as knowledge engineering, knowledge representation, qualitative modelling, database design, language engineering, information integration, information retrieval and extraction, knowledge management and organisation, agent-based system design and e-commerce [13]. Ontologies play also a key role in one of the newest area of interest, the Semantic Web, as confirmed by efforts such as OntoWeb (http://www.ontoweb.org) and DAML (http://www.daml.org).

3.1 Definitions

In each area mentioned before, the term "ontology" can take a different meaning. In some cases this term denotes a set of activities performed following a standardised methodology, such as conceptual analysis and domain modelling, while in some other cases it just indicates a warehouse of vocabulary to solve lexical, semantic and synonym problems. It is thus clear that there is no unique definition of the term "ontology". The meaning of the term ontology changes moving from philosophy to AI. The philosophical account for the term ontology is the branch of metaphysics that deals with the nature of Being, and it can be considered as a particular system of categories ac-

counting for a certain vision of the world [13]. In the remainder of the thesis the word ontology is used as an uncountable noun (which does not have a plural form) when it refers to the philosophical notion of ontology, whereas we will use it as a countable noun when referring to the engineering artifacts defined in AI.

One of the most widely quoted definition of "ontology" was given by Tom Gruber in 1993, who states [2, p. 199]:

> *an ontology is an explicit specification of a conceptualization.*

A useful link to find additional information is
http://www-ksl.stanford.edu/kst/what-is-an-ontology.html
maintained by Gruber. According to Gruber an ontology is a quintuple composed of classes, instances, functions, relationships, axioms. Classes correspond to entities of the domain, instances are the actual objects which are in the domain, functions and relationships relating entities of the domain, and finally axioms constrain the meaning and the use of classes and instances, functions and relationships.

Another definition of ontology, often used in literature, has been given by Nicola Guarino in [13]:

> *an ontology is a set of logical axioms designed to account for the intended meaning of a vocabulary.*

From this definition arises the relationship between a conceptualization and the ontology which specifies and formalize it.

Now I present a list of interesting definitions that other researcher groups have studied. In fact, each research group involved in the ontological field has tried to clarify their view on ontologies and have developed their own definition of ontology. These definitions depend on the purposes for which they have been developed, but, despite some minor differences, they refer to the ontology as a common understanding of a domain, and that implies that it is a repository of vocabulary for the knowledge of a domain. The vocabulary contains both formal and informal definitions. Borst has extended Gruber's definition [14, p. 12]:

> *an ontology is a formal specification of a shared conceptualization.*

Studer and collegues have merged Gruber's and Borst's definition, and have provided an explanation for the terms used [15, p. 185]:

> *An ontology is a formal, explicit specification of a shared conceptualisation. A 'conceptualisation' refers to an abstract model of*

> *some phenomenon in the world by having identified the relevant concepts of that phenomenon. 'Explicit' means that the type of concepts used, and the constraints on their use are explicitly defined. ... 'Formal' refers to the fact that the ontology should be machine-readable. 'Shared' reflects the notion that an ontology captures consensual knowledge, that is it is not private to some individual, but accepted by a group.*

Neches and colleagues developed the following definition about what is defined by an ontology [16, p. 40]:

> *An ontology defines the basic terms and relations comprising the vocabulary of a topic area, as well as the rules for combining terms and relations to define extensions to the vocabulary.*

This definition states clearly that an ontology is not only describing the explicit knowledge about a domain, but also the knowledge that can be inferred.
The relationship between ontologies and knowledge bases has been included in the definition of Bernaras and colleagues [17]; they have defined what a knowledge base provides while designing an ontology:

> *The ontology provides the means for describing explicitly the conceptualisation behind the knowledge represented in a knowledge base.*

Swartout and colleagues [18], on the other hand, have defined the contribution given by an ontology to the design of a knowledge base:

> *An ontology is a hierarchically structured set of terms describing a domain that can be used as a skeletal foundation for a knowledge base.*

Each definition presented above highlights a specific aspect of the role played by ontologies. All definitions share the idea that an ontology provides a description of a particular viewpoint about a domain and that such a description must be explicit, because it states a vocabulary for the domain, which is expressed by a certain degree of formality, and that a group commits to use the vocabulary according to the intended meaning associated with it in order to communicate.

Ontologies, then, can be classified into lightweight and heavyweight ontologies, depending on the degree of formality used to express them. Heavyweight ontologies are those which are provided with axioms, inference mechanisms aimed to equip ontologies with deductive power (e.g., inheritance), and that are characterised by a high degree of formality (e.g., underlying formal semantics). Lightweight ontologies, on the other hand, are those ontologies that define a vocabulary of terms with some specifications of their meaning [19].

3.2 Modelling primitives

Before proceeding by illustrating the different types of ontologies we introduce here the conceptual primitives that can be used when knowledge about a domain is formalised in an ontology. In [2] an ontology is defined as the quintuple:

$$(\mathcal{C}, \mathcal{I}, \mathcal{R}, \mathcal{F}, \mathcal{A})$$

where:

- $\mathcal{C}$ is the set of the *concepts*, that is the set of the abstractions used to describe the objects of the world;
- $\mathcal{I}$ is the set of individuals, that is, the actual objects of the world. The individuals are also called *instances* of the concepts;
- $\mathcal{R}$ is the set of relationships defined on the set $\mathcal{C}$, that is, each $R \in \mathcal{R}$ is an ordered n-ple R= $(C_1 \times C_2 \times \ldots \times C_n)$. For example `subconcept-of` is the pair (C_p, C_c) where C_p is the parent concept and C_c is the child concept;
- $\mathcal{F}$ is the set of functions defined on the set of concepts and that return a concept. That is, each element F is a function F: $(C_1 \times C_2 \times \ldots \times C_{n-1} \mapsto C_n)$. For example, the function `Price-of-flat` is function of the concepts `Year`, `Location` and `Number-of-square-metres`, and returns a concept `Price`, that is `Price-of-flat`: `Year` $\times$ `Location` $\times$ `Number-of-square-metres` $\mapsto$ `Price`;
- $\mathcal{A}$: set of axioms, that is first order logic predicates that constrain the meaning of concepts, relationships and functions.

Some of these components have necessarily to be in an ontology. For example, few ontologies include also instances. The simplest type of ontology is composed by the set of concepts $\mathcal{C}$ and the relationships $\mathcal{R}$ (lightweight ontologies), although this limits the knowledge that can be expressed about a domain. Concepts are also called *classes*. Concepts and instances are usually organised in an *Is-a* hierarchy, which permits, when it is a strict relationship (in the mathematical sense), *inheritance* to be exploited in the structure, that is if A is an ancestor of B (denoted by A $\mapsto$ B) and B $\mapsto$ C then, A $\mapsto$ C. When ontologies include also the content concerning ground individuals and their relationships with the concepts they instantiate, then the notion of inheritance is extended to instances and it is called the *instance relation*. The *Is-a* relationship, also called the *subclass relationship*, is not the only one that can be defined on concepts.

Concepts can be defined in terms of characteristic features describing them, that are called *attributes*. If the concepts are organised in an *Is-a* hierarchy, then the inheritance is extended also to attributes. Attributes are shared by concepts either in their original form or modified in order to give the inheriting class, known also as *subclass*, a more restrictive definition than the one provided by the parent concept. Furthermore other properties can be added to form more specialised concepts.

3.3 Ontologies classification

The ontologies presented in the literature can be classified according to different dimensions, which range from the level of generality of the concepts they describe, to the type of knowledge they model. In this section I present the most commonly used classifications of ontologies. The first dimension that can be used to classify ontology is the level of generality that is used in the description of a domain. It is possible to distinguish the following types of ontologies [13]:

- *Top-level ontologies*: this kind of ontology describes very general concepts or common-sense knowledge such as space, time, matter, object, event, action, etc., which are independent of a particular problem or domain.

- *Domain ontologies*: this kind of ontology describes the vocabulary related to a generic domain such as medicine or physics.

- *Task ontologies*: this kind of ontology describes the vocabulary related to a generic task or activity such as diagnosis or selling.

- *Application ontologies*: this kind of ontology describes concepts depending both on a particular domain and on a particular task. They are often a specialisation of both domain and task ontologies and correspond to the roles played by domain entities when they perform certain activities.

Van Heijst and colleagues propose to classify ontologies according to two dimensions, which are the amount and the type of structure of the conceptualisation and the subject of the conceptualisation [20].

- Amount and type of structure of the conceptualisation: this dimension is mainly concerned with the level of granularity of the conceptualisation and thus can be subdivided into:

  - Terminological Ontologies: these are not strictly speaking ontologies but just lexicons that specify the terminology that is used to represent knowledge in the domain of discourse. They do not represent the semantics of the terms;
  - Information Ontologies: they specify the record structure of databases (for example, database schemata). They provide means to record the basic observations concerning instances of the database, but they do not define the concepts that are instantiated by these instances;
  - Knowledge Modelling Ontologies: they specify conceptualisations of knowledge. They are structurally richer than information ontologies and are often specified according to a particular use of the knowledge they describe.

- Subject of the conceptualisation: this dimension concerns the type of knowledge that is modelled in the ontologies. Four categories are distinguished along this dimension:

  - Application Ontologies: specify those concepts that are necessary in order to model the knowledge required for a specific applications. Usually, application ontologies specialise terms taken from more general ontologies such as the domain and the generic ontologies described below and may extend generic and domain knowledge by representing method and task-specific components. Application ontologies are not reusable, they reuse knowledge which may be modelled in ontology libraries by tuning it for the specific application at hand;

- Domain Ontologies: specify those concepts that are specific of a particular domain. Knowledge engineers draw a line between domain ontologies and domain knowledge, where the former has ontological nature and the latter epistemic. Domain knowledge describes factual situations in certain domains whereas domain ontologies specify the constraints to apply on the structure and the content of the domain knowledge. But the distinction between what is ontological and what is epistemological is quite subtle, and therefore such a line often cannot be drawn too neatly.
- Generic Ontologies: specify concepts that are generic across many fields. Concepts in the domain ontologies may specialise those in the generic ontologies in order to tune them to a particular domain. Generic ontologies correspond to the top-level ontologies in Guarino's classification presented above; they typically define concepts like state, event, process, action, etc.
- Representation Ontologies: explicate conceptualisations underlying knowledge representation formalisms. They provide a representational framework without making claims about the world, because they are meant to be neutral with respect to the world. Domain and generic ontologies are described by means of the primitives given in the representation ontologies.

Ontologies differ also in the degree of formality by which the terms and their meaning are expressed in the ontology, as in [21]. Here, the knowledge expressed in the ontology might be the same, but they differ in the way in which it is expressed.
In discussing how formally ontologies are described, it makes little sense to talk about categories, because the degree of formality is better thought of as a continuum rather than a set of classes.
Nevertheless, we can set four points in this continuum:

- Highly informal: are those ontologies expressed in natural language. Term definitions might be ambiguous due to the inherent ambiguity of natural language;
- Semi-informal: these ontologies are expressed in a restricted and structured form of natural language. Restricting and structuring natural language achieves improvement in clarity and reduction in ambiguity;
- Semi-formal: these are ontologies expressed in artificial languages which are formally defined, such as Ontolingua [22];

- Rigorously formal: these are ontologies whose terms are precisely defined with formal semantics, theorems and proofs of desired properties such as *soundness* and *completeness*.

Along the same line, McGuinness in [23] "classifies" ontologies on the ground of their expressiveness, that is, on the grounds of the information that the ontology needs to express. In fact, depending on the different types of interpretation which are associated with the word *ontology*, we can distinguish between less or more complex notions of ontologies, which may range from a *controlled vocabulary*, to a *glossary*, to reach, at the other end of this spectrum, ontologies which also provide general logical constraints such as disjointness, inverse, part of, etc. The points they distinguish in the spectrum are:

- Controlled vocabularies: a vocabulary is the simplest possible notion of ontology, that is a finite list of terms. A typical example of this category is catalogues. Indeed, catalogues provide terms with an unambiguous interpretation, but nothing more;

- Glossaries: they are a list of terms and their meanings. The meanings are usually expressed in natural language statements that are chiefly aimed at humans. These statements, however, are ambiguous and cannot be used by computer agents;

- Thesauri: add to glossaries the semantics emerging from the definition of the relations between terms, such as the synonym relationship. Typically, they do not provide an explicit hierarchical structure, although this can often be deduced by broader or narrower term specifications. A computer agent can often interpret the relationships defined in a thesaurus univocally;

- Informal *Is-a* hierarchies: this category includes many of the ontologies on the web. These are ontologies where a general notion of generalisation and specialisation is provided although it is not strict subclass hierarchy. A typical example is *Yahoo!*, which provides a small number of top-level categories, but does not provide an explicit hierarchical structure, and its hierarchy is not a strict subclass of the *Is-a* hierarchy. In hierarchies that are not strict *Is-a* hierarchies it is not always the case that an instance of a more specific class is necessarily an instance of a more general class, therefore inheritance (with or without exceptions) cannot be assumed here;

- Formal Is-a hierarchies: these are ontologies where concepts are organised according to a strict subclass hierarchy. Thus, for these ontologies inheritance is always applicable because it is always the case that if a concept C is a superclass of the concept C', then any subclass of C must necessarily be a subclass of C'. These ontologies may include only class names;

- Formal instances: ontologies including formal instance relations are a natural extension of ontologies enforcing a strict hierarchical structure. Indeed, some classification schemes include only class names, as we have pointed out when discussing strict subclass hierarchies. When formal instance relations hold, the ontologies include also the content concerning ground individuals and their relationships with the concepts they instantiate;

- Frames (description of concept properties): these are ontologies whose concepts are described in terms of their characteristic properties. For example, a concept `Book` might be described in terms of features like title, author, publisher, etc. The inclusion of properties in the concept description becomes more interesting when inheritance can be applied to these properties, and thus properties can be specified for a more general concept and be inherited down the hierarchy by more specific concepts;

- Value restriction: these ontologies permit to apply restrictions on the values associated with properties. For example, in describing the concept `Book` we might restrict the value to associate with the property `Author` to be composed of maximum two names. These restrictions are usually to be inherited by the sub-concepts of the concept where they are stated for the first time, which clearly poses a problem when the type of hierarchical relation supported by the ontology is not a strict subclass relation;

- General logical constraints: these ontologies are those with the richest expressiveness. For example, properties might be based on mathematical equations, which use values from other properties, or properties might be expressed as logical statements. This type of ontology is usually written in very expressive ontology languages, such as Ontolingua [22], which permit the specification of first order logic constraints on concepts and their properties.

As a final consideration, it may be important to make clear the difference between an application ontology and a knowledge base. The answer

is related to the purpose of an ontology, which is a *particular* knowledge base, describing facts assumed to be always true by a community of users, in virtue of the agreed-upon meaning of the vocabulary used. A generic knowledge base, instead, may also describe facts and assertions related to a particular state of affairs or a particular epistemic state. Within a generic knowledge base, we can distinguish therefore two components: the ontology (containing state-independent information) and the "core" knowledge base (containing state-dependent information).

3.4 Ontology dynamics – different approaches

With rising importance of knowledge interchange, many industrial and academic applications have adopted ontologies as their conceptual backbone. However, business dynamics and changes in the operating environment often give rise to continuous changes to application requirements that may be fulfilled only by changing the underlying ontologies. This is especially true for WWW and Semantic Web applications, that are based on heterogeneous and highly distributed information resources and therefore need efficient mechanisms to cope with changes in the environment. So over a period of time an ontology needs to be modified to reflect changes in the real world, changes in user's requirements, drawbacks in the initial design, to incorporate additional functionality or to allow for incremental improvement. In fact very seldom an ontology is perfect the first time it is made, and then continues, without change, to be as useful over time as when it was first deployed. The reasons for changes are inherent in the complexity of reality and in the limited ability of humans to cope with this complexity. Thus a small list of possible causes of change may be the following:

- Ontologies often contain "design error" and sometimes do not immediately meet the requirements of its users;
- The environment in which the ontology operates can change unpredictably, thereby invalidating the assumptions that were made when the ontology was built;
- Users' requirements can change after the ontology is initially built, requiring that the existing ontology evolves to meet the new requirements.

Another way to comprehend the causes of change is to refer to the Gruber definition of ontology [2] (*it is a specification of a shared conceptualization of a domain*). Analyzing this definition we can enlighten these causes of change:

1. changes in the domain;
2. changes in the shared conceptualization;
3. changes in the specification.

The first type of change is often occurring and is well known in the database schema versioning area. In this case changes in the domain require changes in the database model. An example of this type of change is the merge of two university departments. As the ontology describes the real world, a change in it requires a modification in the ontology itself.

About the second case, it is important to notice that the shared conceptualization of a domain is not a static specification produced only once in the history, but has to be reached over time. Fensel [24] describes ontologies as dynamic networks of meaning, in which consensus is achieved in a social process of exchanging information and meaning. This view gives a dual role to ontologies in information exchange; they provide consensus that is both a pre-requisite for information exchange and a result of this exchange process. Often the conceptualization can change because of the usage perspective. In fact different tasks may need different views on the domain and consequently a different conceptualization. For example when an ontology is adapted for a new task or domain, the modifications represent changes to the conceptualization. When an ontology about a traffic connections with concepts like roads, bridges, is adapted for a bicycle perspective new concepts have to be added and other have to be modified or removed.

Eventually the specification change is a kind of translation, i.e. a change in the way in which a conceptualization is formally recorded. Therefore this type of change is less interesting because it has a simple solution. In fact translation should retain the semantics, as the specification variants should be equivalent and they only cause syntactic change.

At the end of this overview of the causes of changes, it is simple to understand the fundamental necessity to have methods, tools, framework, environment to face the dynamics aspect of ontologies. Many research groups are involved in the solving of this crucial issue. In the following two approaches are presented.

The first is the *Evolution approach*. It is proposed by [25] and [26] and it is developed at the University of Karlsruhe (Germany). This approach intend to face the dynamics problem in all its complexity and to manage all the consequences that changes require.

The second is the *Versioning approach*. It is proposed by [24] and it is developed at the Free University of Amsterdam (The Netherlands). This

approach, differently from the previous one, try to cope with the complexity of the problem suggesting a system based on versions (or better, on ontology versions).

3.4.1 Evolution approach

The *ontology evolution* [24] is the timely adaptation of the ontology as well as the consistent propagation of changes, because a modification in one part of the ontology may generate subtle inconsistencies in other parts of the same ontology, in the instances, depending ontologies and applications. The ontology evolution is becoming more important nowadays. The major reason for this is the increasing number of ontologies in use and the increasing costs associated with adapting them to changing requirements. Developing ontologies and their applications is expensive, but evolving them is even more expensive. However, even though evolution over time is an essential requirement for useful ontologies, appropriate tools and strategies for enabling and managing evolution are still missing. This level of ontology management is necessary not only for the initial development and maintenance of ontologies, but it is essential during deployment, when scalability, availability, reliability and performance are absolutely critical.

Ontology development is now clear, it is a dynamic process starting with an initial rough ontology, which is later revised, refined and filled in the details [27]. Consequently, an ontology almost certainly should be evolved in order:

- to fix "bugs" in the initial design (corrective maintenance);
- to adapt itself to the changes in the environment (adaptative maintenance);
- to improve itself after it has become operational (perfective maintenance);
- to avoid future changes and to alleviate maintenance (preventive maintenance).

Moreover, ontology evolution has to be supported through the entire lifecycle [28].

The most important problem to face when a change to an ontology occurs, is to ensure the **consistency of the ontology** and all the dependent artifacts. When a change to a consistent ontology is identified, we have to understand how to carry out it in order to maintain its consistent state. The

importance of this problem became more clear if we consider the instances that rely on the ontology and all the applications that depend on it. This issue can be faced diving it in four phases as shown in Figure 3.1.

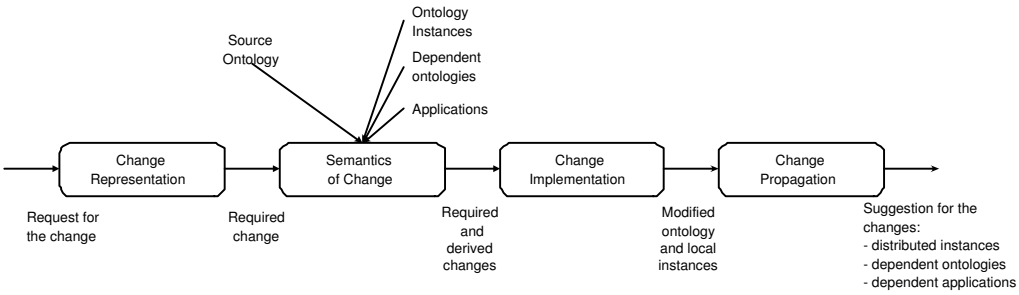

Figure 3.1: Four elementary phases of ontology evolution process

Change Representation

Approaching this problem, the first thing to do is trying to represent the changes, that means to identify and to represent them in the correct format. In the course of ontology evolution different type of changes can be requested. Some of them are simple and interest a defined component of the ontology. Sometimes the changes are complex and involve several components of the ontology, in addition the intent of these changes can be expressed on a higher level. For example, they may need to generate a common superconcept sc of two concepts c_1 and c_2. To take the ontology into the desired state, a list of changes have to be applied:

- Add_Concept sc;
- Delete_SubConceptOf relation from c_1 to its current parent;
- Add_SubConceptOf relation from c_1 to sc;
- Delete_SubConceptOf relation from c_2 to its current parent;
- Add_SubConceptOf relation from c_2 to sc.

This case presents significant drawbacks: (I) five steps are needed to perform the intent change, the process is error prone; (II) a lot of unnecessary changes may be performed if each change is applied alone. For example, removing sub-concept-of relation from c_1 may introduce changes to property instantiations that should be reversed when assign sub-concept-of relation

from c_1 to sc.
To avoid these drawbacks, changes should be express on a more coarse level, with the intent of change directly visible. In this way it become simpler to understand the meaning and the objective of the requested changes. Some example of composite changes are presented in the table 3.1.

Composite change	**Description**
Merge concepts	Replace several concepts with one and aggregate all instances.
Extract subconcepts	Split a concept into a several subconcepts and distribute properties among them.
Extract superconcepts	Create a common superconcepts for a set of unrelated concepts and transfer common properties to it.
Pull up properties	Move properties from a subconcept to superconcept.
Move properties	Move properties from one concept to another concept.
Move instance	Moves an instances from one concept to another.

Table 3.1: Composite changes in the ontology

Semantics of Change

Application of an elementary change in the ontology can induce inconsistencies in other parts of the ontology. We can distinguish *syntax* and *semantic* inconsistency. *Syntax* inconsistency arises when undefined entities at the ontology or instance level are used or ontology model constraints are invalidated. *Semantic* inconsistency arises when the meaning of an entity is changed due to changes performed in the ontology.
For example, removal of a concept which is the only element of domain set for some property results in syntax inconsistency. Resolving that problem is treated as a request for a new change in the ontology, which can induce new problems that cause new changes and so on. Therefore, one change can potentially trigger other changes and cause an unpredictable chain of alterations. If an ontology is large, it may be difficult to fully comprehend the extent and the meaning of each induced change. The task of 'semantics of change' phase is to enable resolution of induced changes in a systematic manner, ensuring consistency of the whole ontology. To help in better understanding the effects of each change, this phase should contribute maximum transparency providing detailed insight into each change being performed.

In the course of evolution, actual meanings of concepts often shifts to better represent the structure of the real world. While some shifts of concept meaning are performed explicitly, a meaning of a concept can sometimes shift implicitly through changes in other parts of the ontology. For example,

consider an ontology describing a relationship between jaguars and persons. In this ontology the meaning of the concept *Jaguar* is clear through the existence of the property *eats* that links *Jaguars* and *Persons* – it is clear that concept *Jaguar* stands for an animal. For any reason one may delete the concept *Person*, which may result in the removal of the property *eats* as well. After this change, the semantics of concept *Jaguar* is not clear any more – is it a Jaguar cat or a Jaguar car? This example highlights the importance of the semantics of change and the problems that arise if we do not treat it properly. For this reason is important to use a rich description of each ontology entity. One way may be attaching meta-information about e.g. essential properties of a concept.

Change Implementation

This phase has the principal aim to prevent undesired changes. Before applying a change to the ontology, a list of all implications to the ontology is generated and presented to the user. He should be able to comprehend the list and approve or cancel the change. When the changes are approved, they are performed by successively resolving changes from the list. At this stage, no action is already done, in fact if changes are cancelled, the ontology remains intact.

Change Propagation

When the ontology is modified, ontology instances need to be changed to preserve consistency with the ontology. This can be performed in three steps. In the first step, the instances are collected in the knowledge base (often they are widespread on the Web). In the second step, modification of instances is performed according to the changes list prepared in the previous phases. In the last step "out-of-date" instances are replaced with corresponding "up-to-date" instances.
Despite this simple process the change propagation arises some important problems. As ontologies often reuse and extend other ontologies, an ontology update might also corrupt ontologies that depend on the modified ontology and also all artifacts that are based on these ontologies. One possible solution to this problem is the recursive application of the ontology evolution process on all the interested ontologies. Nowadays many applications are based on ontologies so when an ontology is modified the application may not work correctly. The ontology evolution system has to recognize which change in the ontology can affect the functionality of the dependent applications and react with the right operations.

Eventually, to explain the complexity of the ontology evolution process Figure 3.2 can be useful. It shows how the process has a cyclic structure,

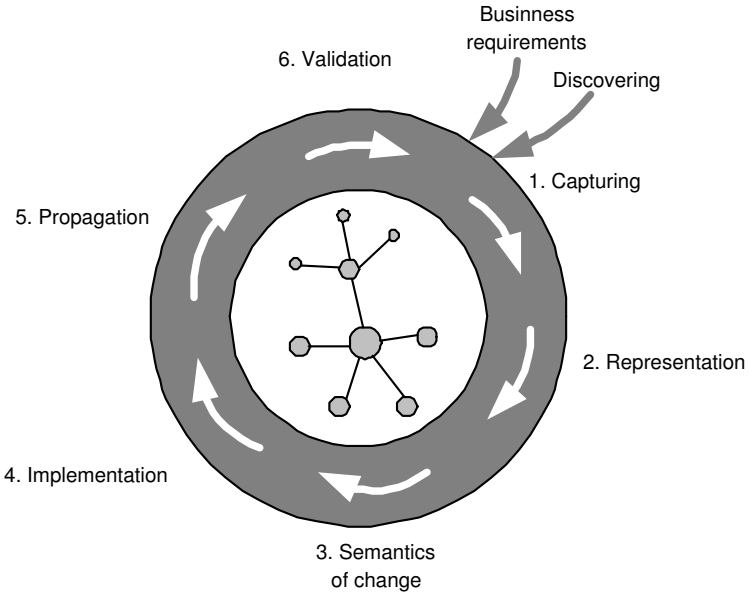

Figure 3.2: Ontology Evolution Process

since validation of realized changes may (automatically) induce new changes in order to obtain a consistent state or to satisfy users' expectations. The phases 2 to 5 correspond to the evolution process explained before; phase 6 gives the user the opportunity to control the process and to supervise it, while phase number 1 is fundamental to continuously identify changes to propose to the user for preventing changes or improving the quality of the ontology.

3.4.2 Versioning approach

In the previous section we have seen one of the most interesting approach to the dynamics of ontologies, but it is not the only one. In literature several studies start assuming that the complexity of the dynamics involving ontologies, applications and instances, is too high to maintain everything aligned. As we have already seen, a request for changing an ontology can start a long process to keep consistent the ontology and all the entities that depend on it. A versioning methodology can handle this complexity providing the user with such a system that manages ontology revisions over time.
Before going deeper in this argument it is better to explain what actually

means *versioning*. In a general sense, ontology versioning means that there are multiple variants of an ontology around. In practice, those variants often originate from changes to an existing variant of the ontology and thus form a derivation tree. It is also possible that different "versions" of ontologies of the same domain are independently developed and do not have a derivation relation. In this case, however, we will not use the word "version", but we will see the variants as separate ontologies that describe the domain from a specific viewpoint or perspective. In this view, ontology versioning is closely related to changes in ontologies [24].

Ontology versioning can be defined as *the ability to handle changes in ontologies by creating and managing different variants of it*. In order to achieve this ability a methodology has been studied. In particular methods to distinguish and recognize versions, and procedures to update and change ontologies are provided.

Analysis of compatibility

The consequences of the change are very important, as we have already said before (see section 3.4.1). Now we discuss the problems that a versioning system has to solve related to the compatibility of changed ontologies and data sources.
A typical scenario presents various versions of many ontologies around. A lot of web pages or applications use (or are intended to use) a specific version of the ontology. At this point incompatibility problems arise. In fact it is not clear which version of an ontology is related to a certain data source. Basically there are tree ways to relate ontology versions with data sources: 1) using the intended version of the ontology; 2) using a newer version of the ontology; 3) using an older version of the ontology. The first type of combination is clearly compatible, so we focus our attention on two directions, which are illustrated in figure 3.3. The terms used in the following are well known in the schema versioning literature; they describe the two directions of investigation.

- **Prospective use**: the use of data sources that conform to a previous version of the ontology via a newer version of the ontology (*i.e. view the data from a newer perspective*).

- **Retrospective use**: the use of data sources that conform to a newer version of the ontology via a previous version of the ontology (*i.e. view the data from an older perspective*).

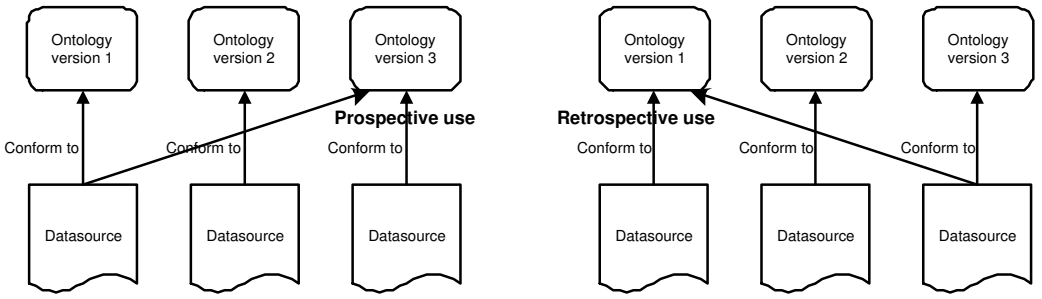

Figure 3.3: Two examples of prospective and retrospective use of ontologies

Based on these directions of use we can now categorize the compatibility revisions into different types. In this categorization, we examine whether it is valid to use a revision of an ontology on the set of all possible instances of an other revision of the ontology. In other words, we assume that the data sources that conforms to a specific version of the ontology uses the whole ontology, i.e., all concepts and relations. This is the worst scenario. Table 3.2 shows the types of compatibility.

		prospective use valid	
		yes	no
retrospective use valid	yes	full compatible	upward compatible
	no	backward compatible	incompatible

Table 3.2: Categorization of compatibility

The types of revisions can be described as follow:

- **full compatible revisions** (upward and backward): the semantics of the ontology is not changed, e.g. syntactic changes or updates of natural language descriptions; this type of change is compatible in both *prospective* use and *retrospective* use;
- **backward compatible revisions**: the semantics of the ontology is changed in such a way that the interpretation of data via the new ontology is the same as when using the previous version of the ontology, e.g. the addition of an independent class; this type of change is compatible in *prospective* use;
- **upward compatible revisions**: the semantics of the ontology is changed in such a way that an older version can be used to interpret newer data sources correctly, e.g. the removal of an independent class; this revision is compatible in *retrospective* use;

- **incompatible revisions**: the semantics of the ontology is changed in such a way that the interpretation of old data sources is invalid, e.g. changing the place in the hierarchy of a class; this type of change is incompatible in both *prospective* use and *retrospective* use.

Note that both backward compatibility and upward compatibility are transitive: when the changes from $v1$ to $v2$ as well as the changes from $v2$ to $v3$ are backward compatible, then the changes from $v1$ to $v3$ are also backward compatible. So, if all subsequent revisions to an ontology up to a certain version are backward compatible, it is also possible to name the resulting version of the *ontology itself* backward compatible. However, it is never allowed to call a version of an ontology *upward* or *full compatible*, because the semantics of future versions are not known beforehand. It is always possible that new versions of ontologies will introduce new things that cannot correctly be interpreted via older versions. Thus, *backward compatibility* can be a characteristic of an ontology, but *upward* or *full compatibility* can not.

Requirements for versioning framework

As we have seen there are various scenarios for ontology changing. Now we presents some requirements and goals for a versioning system.
Principally a versioning methodology should provide mechanisms and techniques to manage changes to ontologies, while achieving maximal interoperability with existing data and applications. This means that it should retains as much information and knowledge as possible, without deriving incorrect information. This general achievement can be translate in a more detailed list of requirements:

- for every use of a concept or a relation, a versioning framework should provide an unambiguous reference to the intended definition (**identification**);

- a versioning framework should make the relation of one version of a concept or relation to other versions of that construct explicit (**change specification**);

- a versioning framework should – as far as possible – automatically translate and relate the versions and data sources, to enable transparent access (**transparent evolution**).

Fensel et al. [24] are working on this framework and till now they have studied in particular the *identification* and *change specification* problems.

Identification

In order to manage various versions of ontologies is fundamental to identify each version unequivocally. About this issue there are different opinions on how to treat changes. In fact one can see an ontology as a specification of a conceptualization, so every modification to the specification can be considered a new conceptualization of the domain. In that case, the descriptions specify different concepts, which are *per definition* not equal. Another perspective can see an ontology primarily as a conceptualization, which is represented as complete as possible in a specification. In this case one could argue that an update to a natural language description of a concept is not a semantic change, but just a refined description of the same conceptualization. To solve this debate an ontology is represented as a file on the Web. Every change that results in a different character representation of an ontology constitutes a revision. When the logical definitions are not changed, the author of the revision has to decide whether the revision is a semantic change and thus forms a new conceptualization with its own identity, or just a change in the representation of the same conceptualization.
The concrete translation of this debate is that an ontology is considered as a resource on the Web, thus it can be identified with a URI (Uniform Resource Identifier). Fensel et al. in this way propose to separate the identity of ontologies completely from the identity of files on the web that specifies the ontology. In other words, the class of ontology resources should be distinguished from the class of file resources. A revision – normally specified in a new file – may constitute a new ontology, but this is not automatic. Every revision is a new file resources and gets a new file identifier, but does not automatically get a new ontology identifier.
The method can be summarized with the following points:

- a distinction between three classes of resources:
  1. files;
  2. ontologies;
  3. lines of backward compatible ontologies.
- a change in a file results in a new file identifier;
- the use of a URL for the file identification;
- only a change in the conceptualization results in a new ontology identifier;

- a new type of URI for ontology identification with a two level numbering scheme:
  - minor numbers for backward compatible modifications (an ontology-URI ending with a minor number identifies a specific ontology);
  - major numbers for incompatible changes (an ontology-URI ending with a major number identifies a line of backward compatible ontologies);
- individual concepts or relations, whose identifier only differs in minor number, are assumed to be equivalent;
- ontologies are referred to by an ontology URI with the according major revision number and the minimal extra commitment, i.e., the lowest necessary minor revision number.

Change specification and transparent evolution

For change specification and transparent evolution, there are two important requirements.
First, in consequences of the suggested practice of referring to the minimal extra addition, the changes in a line of backward compatible ontologies should be easily recognizable and identifiable. A proposal in this direction is to add a class `Addition<Major>.<Minor>`, e.g., `Addition1.2`, of which the new descriptions are an instance. This makes that class a unique identifier for the set of additions in a certain revision. Then, the additions can be retrieved by asking for all instances of a specific "Addition" class.
Second, the relation to a previous version should explicitly be specified. One aspect of this specification is a pointer to the ontology from which it is derived. These pointers form a lattice of versions that can be used to deduce the derivation relation from one version to an arbitrary other version of the ontology. A second aspect of this specification is the relation between concepts and relations in the previous and current version of the ontology. As concepts and relations that do not have the same major number in their identifier are assumed to be different, this specification should both specify equivalence relations as subsumption relations.

Chapter 4

Ontology management – tools and projects

Ontology management is a crucial problem within the Semantic Web. As we have already seen before, ontology evolves at a high frequency rate, so the ontology management is a key challenge to exploit the enormous possibilities of the Semantic Web.

We start by describing SI-Designer, the GUI of the MOMIS project that supports the designer during the source integration phase. We insert its description here because it can be seen as an ontology management tool. In fact, beside its fundamental role during the integration process, it is also used to construct the MOMIS ontology (see sec. 4.1.3).

Secondly we will explain the requirements that an ontology editor should have to be able to handle ontologies during all their life cycle. Doing this we will better understand the problems related to changes and the possible solution to keep the ontology consistent.

Afterwards we will present an overview of the existing tools and projects that aim at helping the user during the ontologies' development and integration process.

4.1 Sources integration with SI-Designer

SI-Designer (**Source Integrator Designer**) *is a designer support tool for semi-automatic integration of heterogeneous sources schemata* (relational, object and semi-structured sources). It has been implemented within the MOMIS project and it carries out integration following a semantic approach which uses intelligent Description Logics-based techniques, clustering techniques and an extended ODMG-ODL language, ODL_{I^3}, to represent schemata

extracted information. Starting from the sources ODL_{I^3} descriptions (local schemata), SI-Designer supports the designer in the creation of an integrated view of all the sources (global schema) which is expressed in the same ODL_{I^3} language.

As described in section 2.2, the integration process is composed of various steps implemented in separate module. All the modules involved are available as CORBA Object and interact using established `idl` interfaces. In particular the SI-Designer provides (Fig 4.1) the designer with a graphical interface to interact with MOMIS modules showing the extracted relationships and helping him in the *Common Thesaurus* construction. Once the *Common Thesaurus* has been built, SI-Designer uses the ARTEMIS [29] module to evaluate a disjoint set of structural similar classes (*clusters*). Each cluster has a corresponding *global class* (a view over all similar classes belonging to the cluster) characterized by a set of global attributes and a *mapping-table*. SI-Designer automatically generates a set of global attributes for each global class and a *mapping table* which maps each global attribute into the local attributes of the classes in the cluster.
Then, a semi-automatic interaction with the designer starts: he may revise the set of global attributes and the mapping table, to assign a name to each global class, so as to achieve a global schema.

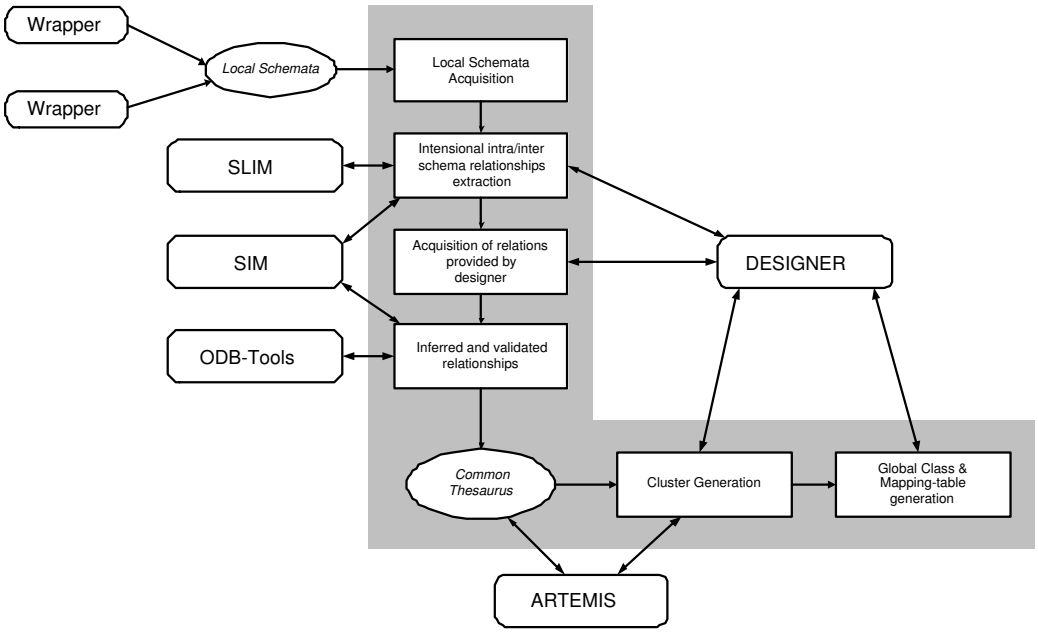

Figure 4.1: SI-Designer Architecture.

The integration process can be divided into two phases:

1. Generation of the Common Thesaurus;
2. Generation of the Global Schema.

At the end of the integration process the Global Virtual View generated can be exported into a XML DTD, by adding the appropriate XML TAGs to represent the mapping table relationships. The use of XML in the definition of the Global Virtual View makes it possible to use MOMIS infrastructure with other open integration information system by the interchange of XML data files. In addition, also the Common Thesaurus is translated into an XML file, so that MOMIS may provides a shared ontology that can be used by different semantic ontology languages.

4.1.1 Generation of the Common Thesaurus

The ***Common Thesaurus*** *is a set of terminological intensional and extensional relationships, describing intra and inter-schema knowledge about classes and attributes of sources schemas*; it provides a reference to define the identification of classes candidate to integration and subsequent derivation of their global representation. In the Common Thesaurus, we express knowledge in form of *intensional relationships* (SYN, BT, NT, and RT) and extensional relationships (SYN_{ext}, BT_{ext}, and NT_{ext}) between classes and/or attribute names. The Common Thesaurus is constructed through an *incremental process* during which relationships are added in the following order:

1. ***Schema-derived relationships***.
   Intensional and extensional relationships holding at intra-schema level. These relationships are extracted analyzing each ODL_{I^3} schema separately. In particular, intra-schema RT relationships are extracted from the specification of foreign keys in relational source schemas or complex attributes (relationships) in object oriented database. When a foreign key is also a primary key both in the original and in the referenced relation, a BT/NT relationship is extracted.

2. ***Lexical-derived relationships***.
   Terminological relationships holding at inter-schema level are extracted by the SLIM module by analyzing different sources ODL_{I^3} schemas together according to the Wordnet supplied ontology.

3. ***Designer-supplied relationships***.
   Intensional and extensional relationships supplied directly by the designer, to capture specific domain knowledge about the source schemata.

This is a crucial operation, because the new relationships are forced to belong to the Common Thesaurus and thus used to generate the global integrated schema. This means that, if a nonsense or wrong relationship is inserted, the subsequent integration process can produce a wrong global schema. Our system helps the designer in detecting wrong relationships by performing a *Relationships validation* step with ODB-Tools. Validation is based on the compatibility of domains associated with attributes. For a complete description see [6].

4. ***Inferred relationships***.
   Intensional and extensional new relationships, holding at intra-schema level, inferred by exploiting inference capabilities of ODB-Tools.

All these relationships are added to the Common Thesaurus and thus considered in the subsequent phase of construction of Global Schema.
Relationships defined in each step hold at the intensional level by definition. Furthermore, in each of the above steps the designer may "strengthen" a terminological relationships SYN, BT and NT between two classes C1 and C2, by establishing that they hold also at the extensional level, thus defining also an extensional relationship.

The WordNet lexical system

We use WordNet [9] lexical system as *lexicon reference ontology*; this choice is due most to these reasons: WordNet is a well-known and widespread lexical database, it seems to be a complete and professional instrument, is mentioned in many scientific papers and it is continuously reviewed (for more details see [6]).
Having different sources, it is however necessary to translate all their attributes and classes' names in a common language, thus resulting in a mapping process of all those terms in a well defined set of concepts (assumed as the WordNet ontology). This mapping process is often referred to in literature as the *sources annotation phase*.

Sources annotation phase

What exactly the designer is requested for annotating the schema is to manually choose a WordNet's meaning for each schema element. Starting from the schema to be integrated, the designer must declare a relation between each term (class or attribute name) and the WordNet meaning appropriated for the particular context. Then the designer must choose one or more meanings for each name. This is a two-step process.

Word form choice In this step, the WordNet morphologic processor should aid the designer. A *word form* is the word without any suffixes due to declination or conjugation. If such *word form* is not found or there is ambiguity (E.g. `axes` has 3 word forms: `ax` (1 sense), `axis` (5 senses), `axe` (2 senses)), or it is not satisfactory, the designer can set a custom *word form*. Notice that this operation, however, does not allow the designer to physically extend WordNet ontology with his particular annotation concept.
About this issue a recent study to extend WordNet has been conducted by V. Guidetti in her thesis [30].

Meaning choice The designer can choose to map an element on zero, one or more senses. For example WordNet has 15 meanings for the *word form* `address` from which the most appropriate ones to the particular context are chosen. Notice that the user can choose a sense only among the existing ones, that is he is not able to extend WordNet with his new meanings which could be more appropriate to the particular domain of interest than the pre-existing ones.

The relations coming from WordNet are proposed as semantic relations to be added to the *Common Thesaurus* according to the following mapping:

Synonymy:	corresponds to a	SYN	relation
Hypernymy:	corresponds to a	BT	relation
Hyponymy:	corresponds to a	NT	relation
Holonomy:	corresponds to a	RT	relation
Meronymy:	corresponds to a	RT	relation
Correlation:	corresponds to a	RT	relation

4.1.2 Generation of the Global Schema

Here we describe the process to construct the global ontology, that is, the construction of the Global Virtual integrated View (GVV) of the managed sources, its annotation and the mapping description between the GVV itself and the integrated sources.

Once the Common Thesaurus has been built, SI-Designer can generate the global class. Such activity is carried out in the following way:

1. *Cluster generation*;

2. *Global class generation.*

Cluster generation

To integrate the ODL_{I^3} classes of the different sources into a global ODL_{I^3} class, MOMIS employs ***hierarchical clustering techniques*** based on the concept of ***affinity***. In this way, MOMIS identifies ODL_{I^3}classes that describe the same or semantically related information in different source schemas and give a measure of the level of matching of their structure. This activity is performed by **ARTEMIS** [10], evaluating a set of ***affinity coefficients*** for all possible pairs of ODL_{I^3} classes on the basis of the relationships of the common thesaurus. Affinity coefficients determine the degree of semantic relationship between two classes based on their names (***name affinity coefficient***) and on their attribute refined by the data type validation (***structural affinity coefficient***). A comprehensive affinity value, called ***global affinity coefficient***, is the linear combination of the name and structural affinity coefficients. *The output of the clustering procedure is an* ***affinity tree****. The leaves represent all the local classes*: adjacent leaves represent classes with high affinity, while leaves far apart from each other represent classes with low affinity; *each node represents a clustering level and is associated to the affinity coefficient between the sub-trees (clusters) it joins.* Within SI-Designer, the designer, at any iteration, can insert a ***threshold value*** which is used by ARTEMIS to build *clusters*: *each cluster is made of all the classes belonging to a sub-tree having a coefficient at the root node which is higher than the threshold value.*

Global class generation

Starting from the output of the cluster generation, we define, for each cluster, a ***Global Class*** that represents the mediated view of all the classes of the cluster.

For each global class, SI-Designer creates a set of ***global attributes*** and, for each of them, it determines the correspondence with the local attributes (i.e. those of the classes belonging to the cluster to which the global class corresponds). In some cases, the correspondence is unique while in other cases the tool identifies different kinds of correspondences but cant solve their ambiguity: in this case the tool asks the designer to choose the right one. The tool builds the global attributes set to be associated to a cluster in two phases: (1) *Union* of the attributes of all the classes belonging to the cluster; (2) *Fusion* of the "similar" attributes. In this phase SI-Designer tries to eliminate redundancies considering the relationships of the Common

Thesaurus. The fusion process is automatic for the attributes which are associated by validated relationships while it is not always automatic when their relationships are not validated.

The attributes having a *valid terminological relationship* are unified into a unique global attribute. The attribute unification process is performed automatically for what concerns names according to the following rules: for attributes that have a SYN relationship, only one term is selected as the name for the corresponding global attribute; for attributes that have a BT/NT relationship, a name which is a broader term for all of them is selected and assigned to the corresponding global attribute.

For each global class a persistent ***mapping-table*** storing all the intensional mappings is generated. The mapping-table is a table whose columns represent the set of the local classes which belong to the cluster and whose rows represent the global attributes.

An element `MT[ag][L]` represents how the global attribute `ag` is mapped into the local class `L`. Each element `MT[ag][L]` of the table has to assume one of the following values:

- `MT[ag][L] = al`: the global attribute `ag` maps into the al local attribute.
- `MT[ag][L] =` $\texttt{al}_1$ `and` $\texttt{al}_2$ `and ...and` $\texttt{al}_n$: this correspondence is used when the value of the `ag` attribute is the concatenation of the value assumed by a set of attributes $\texttt{al}_i$ belonging to the same local class.
- `MT[ag][L] = case of al const`$_1$ `:` $\texttt{al}_1$`, ...constn :` $\texttt{al}_n$: this situation occurs when the `ag` global attribute can assume one value in a set of $\texttt{al}_i$ belonging to the same local class `L` and the chosen value depends on a third attribute, `al`, which acts as a selector.
- `MT[ag][L] = const`: in this case a global attribute value does not refer to any local attribute and a constant value is set by the designer.
- `MT[ag][L] = null`: in this case no attribute of the class `L` corresponds to the global attribute `ag`.

4.1.3 The MOMIS ontology

At this point we can precisely define MOMIS ontology constructed at the end of the presented process. We can identify the following components:

- ***local ontologies of the sources***: a local ontology is a formal explicit description of concepts in an information source (*classes*), properties of each concept describing various features of the concept (*attributes*), and restrictions on instances of classes (*integrity constraints*).
  Since this description is expressed by a formal language, the local ontology is called also ***local schema***.

- ***annotation of the local sources***: having different sources, it is necessary to analyze the meaning of the terms in different local ontologies with respect to a common *lexicon ontology* as Wordnet, that resulting in a mapping of all those terms in a well defined set of concepts of the adopted lexicon ontology.

- ***Common Thesaurus***: correlations among the terms coming from different local ontologies; in particular it is a set of terminological intensional and extensional relationships, describing intra and inter-schema knowledge about classes and attributes of sources' schemas;

- ***global ontology***: it is a *global integrated view* of the managed local ontologies and a mapping between the global integrated view itself and the integrated local ontologies.
  Since the global integrated view is expressed by a formal language (the same language used to express a local ontology), the global ontology is called also ***global schema***.

- ***annotation of the integrated view***: meaning of the terms (classes and attributes) of the global ontology with respect to a common lexicon.

It is important to note that the last component is new with respect to the present MOMIS system; it has been conceived during the study of this thesis. We will explain how to annotate the integrated view in the next chapter.

The building of the MOMIS ontology, as we have just seen, is an incremental process that starts with the acquisition of the sources' schema, it continues with the extraction of the integrity constraint rules and of the inter/intra schema relationships terminating with the building of the Global Virtual View and the definitions of the mapping rules among the global classes and the local interfaces. Furthermore we add the annotation of the integrated view in order to enrich its semantics and expressive power and preparing the schema to the integration with new sources. We will describe this issue in details in the next chapter.

We recall here the fundamental role of ontology to elaborate the global view, in fact, the cluster generation (that is the core phase of the process)

relies mainly on the exploitation of the knowledge represented by the ontology.

Eventually the MOMIS ontology can be considered as an *Application Ontology*, with respect to the classification given in section 3.3, because it describes concepts strictly related to a particular domain and dependent on the specific task they are used for.

4.2 Requirements for ontologies editors

Since an ontology is usually developed using an ontology editor, many requirements for the ontology evolution have to be part of the ontology editors. An ontology editor must provide an interface that allows the knowledge engineer to modify the underlying ontology. The interface is based on the set of available ontology changes. Moreover, there are many features which can significantly improve the usability of an ontology editor and enhance its functionality regarding the ontology evolution.

Following [25] we can divide these requirements in several groups:

- *Functional requirement* specifies all evolution changes that must be supported;
- *User's supervision requirement* enables the user-driven process of change resolving;
- *Transparency requirement* deals with providing control of the evolution process through an insight into the scope of an evolution operation before the operation is applied;
- *Reversibility requirement* states how the effect of evolution changes can be undone;
- *Auditing requirement* is related to the management of the ontology change history;
- *Ontology refinement requirement* provides support for continual ontology improvement;
- *Usability requirement* allows the user to manage changes more easily by finding ontology inconsistencies and providing the explanation to solve them.

Now a more detailed explanation of these requirements will be presented. In this way we can improve the comprehension of the complexity that an ontology editor have to face.

It is important to note that it is very difficult to find an editor that accomplish to all these requirements, because of the high level of complexity and completeness of such an editor, but it can be useful to understand the "ideal" characteristics of an editor if we want to evaluate easily the "real" tools we will present later in section 4.3.

Functional requirement

The functional requirement specifies which functionality must be provided for the *ontology development and evolution*. This functionality heavily depends on the underlying *ontology model*. The more powerful and expressive model requires a richer set of modelling primitives. Thus, before speaking about functional requirements, the notion of an ontology itself has to be clarified. As we have already seen there is not a standard ontology model, but there are several attempts to achieve this objective.
Due to differences in ontology models, we concentrate on the "common" features of ontology models, namely *concepts*, *properties*, *instances*, as well as *concept inheritance*. Each of these ontology entities can be updated by one of the meta-change transformations: *add*, *remove*, *modify*.
The existence of the "modify" change causes the set of primitives not to be minimal with respect to completeness. However, this change adds some important semantic variants to the set of changes, since a modify change is not equivalent to a removal followed by an addition. The difference is that the modification of an entity (i.e. renaming a concept) maintains its identity, while removing and adding loses its identity.
The previous changes are called *elementary* changes since they cannot be decomposed into simpler changes. Elementary changes in the ontology specify fine-grained changes that can be performed in the course of ontology evolution. However, this granularity is not always appropriate. Often, the *intent* of the changes may be expressed on a higher level. *Composite* changes specify coarse-grained changes that can be performed to improve the ontology structure according to some criteria. They are more powerful, since the designer does not need to go through every step of the sequence of basic changes to achieve the desired effect. Moreover, composite changes often have more meaningful semantics. For example, the semantics of moving the concept from one parent concept to another is clearly different from the semantics of removal and addition of a `subConceptOf` relation. While "move" as a composite change maintains the identifiers of a `subConceptOf` relation and

preserves all properties and instances, the removal and subsequent addition create a new identifier for a `subConceptOf` relation and cause the loss of much information (e.g. at the instance level). Changes are applied to an ontology in a valid state, and after all changes are performed, the ontology and dependent artifacts must evolve to another valid state.

User's supervision requirement

The ontology evolution is a process of changing an ontology while *maintaining its consistency*. However, there are many ways to achieve consistency after a change request. For example, when a concept from the middle of the hierarchy has been deleted, all subconcepts may either be deleted or reconnected to other concepts. If subconcepts are preserved, then properties of the deleted concept may be propagated, its instances distributed, etc. Thus, *for each change in the ontology, it is possible to generate different sets of additional changes, leading to different final consistent states.*
Hence, a mechanism is required for users to manage changes resulting not in an arbitrary consistent state, but in a consistent state fulfilling the users preferences. In order to enable the user to obtain the ontology most suitable to her needs, an ontology editor should allow the customization of the ontology evolution process. One mean is to enable the user to set up one of *evolution strategies* that are used for resolving the changes.
An evolution strategy unambiguously defines the way how elementary and composite changes will be resolved. Typically, a particular evolution strategy is chosen by the user at the start of the ontology evolution process. Thus, an evolution strategy defining a common policy must be chosen to specify how to handle each of the following situations:

- how to handle orphaned concepts - those concepts that don't have parents any more;
- how to handle orphaned properties - those properties that don't have parents any more;
- how to propagate properties to the concept whose parent changes;
- what constitutes a valid domain of a property;
- what constitutes a valid range of a property;
- whether a domain (range) of a property can contain a concept that is at the same time a subconcept of some other domain (range) concept;
- the allowed shape of the concept hierarchy;

- the allowed shape of the property hierarchy;

For each of these situations, there is a set of possible options, e.g. in case of the first issue, orphaned subconcepts of a concept may be connected to the parent concept(s) of that concept, connected to the root concept of the hierarchy or deleted as well.

Transparency requirement

A change in one part of an ontology may have far reaching *consequences* on other parts of the ontology and associated instances. If an ontology is large, it may be difficult to fully comprehend the extent and meaning of each change. To improve understanding of effects of each change, the ontology evolution should provide maximum transparency into details of each change being performed. Transparency should provide a human-computer interaction for evolution by presenting change information in an orderly way, allowing easy spotting of potential problems and alleviating the understanding of the scope of the change.
Before any change is applied to the ontology, a list of all implications must be generated and reported to the user. The ontology engineer should approve or cancel the changes. If the changes are cancelled, the ontology should remain intact.

Reversibility requirement

There are numerous circumstances where it may be desired to reverse the effects of changes. The reversibility requirement states that an ontology editor has to allow *undoing changes* at the users request. Consequently, the user can control changes and make appropriate decisions.
It is important to note that reversibility means undoing all effects of a change, which may not be the same as simply requesting an inverse change manually. For example, if a concept is removed from a concept hierarchy, its subconcepts will be modified (e.g. attached to the root). Reversing such change is not equal to recreating the deleted concept one also needs to revert the concept hierarchy into an original state.

Auditing requirement

Often business applications use ontologies, so the need for auditing ontology evolution become more important and changes to business information are accompanied with responsibility for their effects on the business. Auditing is therefore a typical component of business systems, and must be reflected

in the ontology evolution as well.
The ontology evolution auditing involves the following aspects:

- Keeping a detailed log of all performed changes allowing later reconstruction of the events that led to the current state of the ontology;
- Associating meta-information with each log change, such as textual change description, cost of change, time of change etc.;
- Tracking the identity of the change author.

The auditing requirement is also related to the reversibility requirement, since the auditing log is typically used to provide reversibility. The auditing log can also serve as a source for information mining about change trends.

Ontology refinement requirement

This requirement states that potential changes improving the ontology may be discovered semi-automatically from the ontology-based data and through the analysis of the users behaviour. We distinguish (i) *structure-driven*, (ii) *data-driven* and (iii) *usage-driven* change discovery.

(i) The *structure-driven change discovery* identifies some heuristics to improve an ontology, based on the analysis of the structure of the ontology: for example, if all subconcepts have the same property, the property may be moved to the parent concept or if a concept has a single subconcept it should be merged with its subconcept. A concept without properties is a candidate for deletion

(ii) The *data-driven change discovery* states that some changes are implicit changes in the domain, reflected in its instances and can be discovered only through their analysis. A set of heuristics is necessary, for example: a concept with no instances may probably be deleted; a concept with many instances is a candidate for being split into subconcepts and its instances distributed among newly generated concepts.

(iii) The *usagedriven ontology evolution* takes into account the usage of the ontology in the knowledge management system. It is based on the analysis of the users behaviour in two phases of a knowledge management cycle: in providing knowledge by analysing the quality of annotations, and in searching for knowledge by analysing the users queries and the responses from the knowledge repository. For example, by tracking when the concept was last retrieved by a query, it may be possible to discover that some concepts are out of date and should be deleted or updated.

Usability requirement

An ontology editor has to have an *interface* that enables the user to create and maintain ontologies, one that is easily understood and allows the user to work efficiently with all the complexities inherent in an ontology editor. An ontology editor has to guide the user through the ontology development process by providing additional information, such as why the users activity did not succeed, or what else he has to do in order to finish the current activity. Moreover, a good ontology editor must provide capabilities for identifying inconsistencies. When such conflicts arise, an editor must assist the user in identifying the source of the problem and resolving it.

4.3 Comparison of ontologies tools

In the last years, a high number of environments for ontology construction and ontology use have been conceived. Tool support is really important both for the ontology development process (ontology building, annotation, merge, etc.) and for the ontology usage in applications, such as electronic commerce, knowledge management, the Semantic Web, etc.

Tools can be divided into two big categories according to [31]:

Ontology development tools. This group include tools, environments and suites that can be used for building a new ontology from scratch or reusing existing ontologies. Apart from the common edition and browsing functionality, these tools usually include ontology documentation, ontology exportation and importation from different formats, graphical views of the ontologies built, ontology libraries, attached inference engines, etc.

Ontology merge and integration tools. These tools have been conceived to solve the problem of merging or integrating different ontologies on the same domain. This need becomes apparent when two companies or organizations are merged together, or when it is necessary to obtain a better ontology from other existing ontologies in the same domain.

Now we present some of the most important tools following the work done by the members of the Ontology-environments SIG in the OntoWeb project [31]. Afterwards some critical comments are reported (page 61 and 68).

4.3.1 Ontology development tools

In this section we will present the most important tools that can be used to build an ontology from scratch or to reuse other existing ontologies. A brief

description of each tool will be provided. Information about its main features and functionalities and its relationship with KR (knowledge representation) formalisms will be illustrated.

OILEd

OILEd is a graphical ontology editor developed by the University of Manchester that allows the user to build ontologies using DAML+OIL.

The knowledge model of OILEd is based on that of DAML+OIL, although this is extended by the use of a frame-like presentation for modelling. Thus OILEd offers a familiar frame-like paradigm for modelling while still supporting the rich expressiveness of DAML+OIL where required. Classes are defined in terms of their superclasses and property restrictions, with additional axioms capturing further relationships such as disjointness. The expressive knowledge model allows the use of complex composite descriptions as role fillers.

The main task that OILEd is targeted at is that of editing ontologies or schemas, as opposed to knowledge acquisition or the construction of large knowledge bases of instances. Although functionality is provided that allows the definition of individuals, this is primarily intended for the definition of nominal, which are used in the DAML+OIL one-of construction.

A key aspect of OILEd's behaviour is the use of the FaCT reasoner to classify ontologies and check consistency via a translation from DAML+OIL to the SHIQ description logic. This allows the user to describe their ontology classes and have the reasoner determine the appropriate place in the hierarchy for the definition.

The DAML+OIL RDF Schema is used for loading and storing ontologies. In addition, the tool will read and write concept hierarchies in pure RDF and will render ontology definitions as HTML for browsing and as SHIQ for later classification by the FaCT reasoner.

OILEd version 3.4 is implemented in Java and is freely available from the OILEd web site. Further information and relevant publications are also available at the web site.

URL: http://oiled.man.ac.uk/

OntoEdit

OntoEdit is an Ontology Engineering Environment supporting the development and maintenance of ontologies by using graphical means. OntoEdit has

been developed by OntoPrise GmBH [1].

OntoEdit is built on top of a powerful internal ontology model. This paradigm supports representation-language neutral modelling as much as possible for concepts, relations and axioms. Several graphical views onto the structures contained in the ontology support modelling the different phases of the ontology engineering cycle.

The tool allows the user to edit a hierarchy of concepts or classes. These concepts may be abstract or concrete, which indicates whether or not it is allowed to make direct instances of the concept. A concept may have several names, which essentially is a way to define synonyms for that concept. As it happens in the well-known "copy-and-paste" functionality, the tool permits the reorganizing of concepts within the hierarchy.

The tool is based on a flexible plugin framework. Firstly this easily makes it possible to extend functionality in a modularized way. The plugin interface is open to third parties which enables users to extend OntoEdit easily by additionally needed functionalities. Secondly, having a set of plugins available like e.g. a domain lexicon, an inferencing plugin and several export and import plugins, this allows for user-friendly customization to adapt the tool to different usage scenarios.

Currently the version 2.6 is available. The Professional Version of OntoEdit contains many more plugins than the free version. Beside others the functionality is extended by (i) an inferencing plugin for consistency checking, classification and execution of rules, (ii) collaborative engineering of ontologies and (iii) an ontology server for administration of ontology libraries, collaborative sharing of ontologies and a persistent storage for ontologies.

OntoEdit web portal contains tutorials, technical documents, version details, plug-ins and there is the possibility to download a free version and have information and costs of professional version.

URL: http://www.ontoprise.de/ontoedit_en.htm

Protégé-2000

Protégé-2000 is the latest tool in an established line of tools developed at Stanford University for knowledge acquisition. Protégé- 2000 is freely available for download under the Mozilla open-source license from the web site.

[1] http://www.ontoprise.de/

Protégé-2000 provides a graphical and interactive ontology-design and knowledge-base development environment. It helps knowledge engineers and domain experts to perform knowledge-management tasks. Ontology developers can have quick access to relevant information whenever they need it, and can use direct manipulation to navigate and manage an ontology. Tree controls permits quick and simple navigation through a class hierarchy. Protégé uses forms as the interface for filling in slot values.

The *knowledge model* of Protégé-2000 is OKBC-compatible. It includes support for classes and the class hierarchy with multiple inheritance; template and own slots; specification of pre-defined and arbitrary facets for slots, which include allowed values, cardinality restrictions, default values, and inverse slots; metaclasses and metaclass hierarchy.

One of the major advantages of the Protégé-2000 architecture is that the system is constructed in an open, modular fashion. Its component-based architecture enables system builders to add new functionality by creating appropriate plugins. The Protégé Plugin Library (http://protege.stanford.edu/plugins.html) contains contributions from developers all over the world.

Most plugins fall into one of the three categories: (1) *backends* that enable users to store and import knowledge bases in various formats; (2) *slot widgets*, which are used to display and edit slot values or their combination in a domain-specific and task-specific ways, and (3) *tab plugins*, which are knowledge-based applications usually tightly linked with Protégé knowledge bases.

Current *backend plugins* (and standard backends) include support for storing and importing ontologies in RDF Schema, XML files with a DTD, and XML Schema files. The development of DAML+OIL support is in its final stages. Available *slot widgets* include user-interface components to display GIF images, as well as video and audio. A diagram widget allows developers to build elements of a knowledge base by drawing a diagram in which nodes and edges are themselves frames of particular types (distinguished by shape and color).

Currently available *tab plugins* provide capabilities for advanced visualization, ontology merging and version management, inferencing, and so on.

The OntoViz and Jambalaya tabs, for example, present different graphical views of a knowledge base, with the Jambalaya tab allowing interactive navigation, zooming in on particular elements in the structure, and different layouts of nodes in a graph to highlight connections between clusters of data.

Some tabs provide support to express constraints on the data. Other tabs (for instance PAL) provide inference engine capable to analyze data and to tell the users which constraints the instances in the knowledge base violate

and how.

The PROMPT tab provides an environment for managing multiple ontologies (see also next section). Its components include tools for ontology merging that help the user to find similarities between source ontologies and to merge the ontologies; for ontology versioning, which automatically finds a structural "diff" between versions of an ontology; and for extracting semantically complete subparts of an ontology and rearranging frames in different linked ontologies.

The UMLS and WordNet tabs enable users to import and integrate elements of the large on-line knowledge sources into their ontologies.

The project portal is useful to find documentations, ontologies, tutorials and to download the tool and the plug-ins.

URL: http://protege.stanford.edu/

WebODE

WebODE is an ontological engineering workbench that provides varied ontology related services and covers and gives support to most of the activities involved in the ontology development process and in the ontology usage. It is built on an application server basis, which provides high extensibility and usability by allowing the easy addition of new services and the use of existing services. It has been developed by the Ontology & Knowledge Reuse Group (AI Lab), from the Artificial Intelligence Department of the Computer Science Faculty (FI) from the Technical University of Madrid (UPM).

WebODE ontologies are represented using a very expressive *knowledge model*, based on the reference set of intermediate representations of the *METHONTOLOGY* methodology, which includes ontology components such as concepts (with instance and class attributes), partitions, adhoc binary relations, predefined relations (taxonomic and part-of ones), instances, axioms, rules, constants and bibliographic references. It also allows the importation of terms from other ontologies, through the use of imported terms.

Ontologies in WebODE are stored in a relational database. Moreover, WebODE provides a well-defined service-oriented API for ontology access that makes easy the integration with other systems. Ontologies built with WebODE can be easily integrated with other systems by using its automatic *exportation* and importation services from and into XML, its translation services into and from varied ontology specification languages (currently,

RDF(S), OIL, DAML+OIL, CARIN and FLogic), and its translation services to other languages and systems, such as Java.

Ontology edition in the WebODE ontology editor is aided both by form based and graphical user interfaces, a user-defined views manager, a *consistency checker*, an *inference engine*, an *axiom builder* and the *documentation service*. Two interesting and novel features of WebODE with respect to other ontology engineering tools are: instance sets, which allow to instantiate the same conceptual model for different scenarios, and conceptual views from the same conceptual model, which make it possible to create and store different parts of the ontology, highlighting and/or customizing the visualization of the ontology for each user.

The graphical user interface make it possible to browse all the relationships defined on the ontology as well as graphical-pruning these views with respect to selected types of relationships. Mathematical properties such as reflexive, symmetric, etc. and other user-defined properties can also be attached to the "ad hoc" relationships.

The collaborative edition of ontologies is ensured by a mechanism that allows users to establish the type of access of the ontologies developed, through the notion of groups of users. *Synchronization mechanisms* also exist that allow several users to edit the same ontology without errors.

Constraint checking capabilities are also provided for type constraints, numerical values constraints, cardinality constraints and taxonomic consistency verification (i.e., common instances of disjoint classes, loops, etc.).

Finally, WebODEs inference service has been developed in Ciao Prolog. A subset of the OKBC primitives has been defined in prolog for their use in this inference engine. Additionally, the WebODE Axiom Builder transforms first-order logic axioms and rules into Prolog, if possible, so that they can be used in it as well.

On the web site is possible to find documents related to the project.

URL: http://delicias.dia.fi.upm.es/webODE/

Comments

Now some comments and observations related to the previous tools will be presented.

All editors permit elementary changes, but only OntoEdit provides support for some composite changes with its functionality "copy-and-paste".

Most of the existing systems provide only one possibility for realizing a change, and this is usually the simplest one. It means that users are not able to control the way changes are performed (supervision). Moreover, users hardly obtain explanations why a particular change is necessary (transparency). Furthermore, there is no possiblity to undo effects of changes (reversibility). None of the existing development systems offer support for semi-automatic ontology improvement.

About software architecture and platform, most of the tools are moving towards Java platforms, and most of them already have an extensible architecture. Some of them are available open source and can be extended by the users (OILEd, Protégé2000, OntoEdit). In certain cases functionalities can be added with plugins (Protégé2000, OntoEdit). Only OntoEdit prof., Protégé2000 and WebODE store ontologies in databases, while other store ontologies in files.

Almost all tools have a good capability to import and to export ontologies from and to many different languages: XML, OIL, RDF(S), DAML+OIL are the most important.

Otherwise there are two families of tools from the KR paradigm: description-logic based tools, such as OILEd; and the rest of tools, which permit to represente knowledge following a hybrid approach based on frames and first order logic.

An important aspect to observe is the inference service provided by each tool. In particular we mean built-in inference engines, constraint and consistency checking mechanisms, automatic classification and exception handling. OntoEdit prof. uses OntoBroker, OILEd performs inference using the FACT inference engine, Protégé2000 and WebODE use internal inference engine, PAL and Prolog respectively.

WebODE has interesting functionality to permit the collaborative edition of ontologies and a synchronization mechanism to avoid errors.

As regards usability, each tool tries to provide special functionality to help the user with the visualization of the ontologies or the navigation, or some zooming options, or customization.

4.3.2 Ontology merge and integration tools

In this section we will present the most important tools that can be used to merge or to integrate two or more ontologies. A brief description of each tool will be provided. Information about its main features and functionalities will be illustrated.

Chimaera

Chimaera is a merging and diagnostic web-based browser ontology environment developed by the Knowledge Systems Laboratory at Stanford University.

Chimaera is built on a platform that handles any OKBC compliant representation system. Chimaera accepts over 15 designated input format choices (such as ANSI KIF, Ontolingua, Protégé, CLASSIC, XOL, etc.) as well as any other OKBC compliant form. It will soon be compliant with other emerging standards such as RDF and DAML.

Chimaera contains a simple editing environment in the tool and also allows the user to use the full Ontolingua editor/browser environment for more extensive editing. Ontolingua is not a requirement however; other editors could be used in its place. It facilitates *merging* by allowing users to upload existing ontologies into a new workspace (or into an existing ontology). Chimaera will suggest *potential merging candidates* based on a number of properties. It generates a name resolution list that may be used as a guide through the merging task. The user sees a display of the places where the two terms appear in the hierarchy (with only the connected portions of the hierarchy displayed). The user may browse the hierarchy in more details by doing things like expanding subclasses. The user may also view the definitions of the terms and, within Ontolingua, also obtain the results of similar and different structural comparisons of the definitions. The user may then choose to merge the terms with a simple menu choice from the class menu. Chimaera allows the user to choose the level of vigor with which it suggests merging candidates. Higher settings, for example will look for things like possible acronym expansion (which was extremely valuable in the use of Chimaera on some government knowledge bases).

Chimaera also supports a taxonomy resolution mode. It looks for a number of syntactic term relationships (such as $< X - Y >$ and $< Y >$ since the two are usually subclass related). When attached to a classifier, it can look for semantic subsumption relationships as well.

Chimaera includes an analysis capability that allows users to run a diagnostic suite of tests selectively or in their entirety. The output is displayed as an interactive log that allows users to see the results of the tests and also to explore the results. The tests include incompleteness tests, syntactic checks, taxonomic analysis, and semantic checks. This is essentially a "to-do" list containing updates that would likely need to be done before the ontologies would be of the most use. The list contains things such as terms that are used but that are not defined, terms that have contradictory ranges, cycles

detected in the ontology definitions.

On Chimaera web site it is possible to find documents and tutorials. Moreover you can demo Chimaera on line and find fully functional version at http://www-ksl-svc.stanford.edu/ after the login.

URL: http://www.ksl.stanford.edu/software/chimaera/

PROMPT

PROMPT is a tool for semi-automatic guided ontology merging. It is a plugin for Protégé-2000 (see page 4.3.1).

PROMPT leads the user through the ontology-merging process, identifying possible *points of integration*, and *making suggestions* regarding what operations should be done next, what *conflicts* need to be resolved, and how those conflicts can be resolved. PROMPT's ontology-merging process is interactive. A user makes many of the decisions, and PROMPT either performs additional actions automatically based on the user's choices or creates a new set of suggestions and identifies additional conflicts among the input ontologies.

The tool takes into account different features in the source ontologies to make *suggestions* and to look for *conflicts*. These features include:

- names of classes and slots (e.g., if frames have similar names and the same type, then they are good candidates for merging)
- class hierarchy (e.g., if the user merges two classes and PROMPT has already thought that their superclasses were similar, it will have more confidence in that suggestion, since these superclasses play the same role of the classes that the user declared as the same)
- slot attachment to classes (e.g., if two slots from different ontologies are attached to a merged class and their names, facets, and facet values are similar, these slots are candidates for merging)
- facets and facet values (e.g., if a user merges two slots, then their range restrictions are good candidates for merging)

In addition to providing suggestions to the user, PROMPT identifies conflicts. Some of the *conflicts* identified by PROMPT are:

- 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.

The features in the list above employ the graph structure of the ontologies to a limited extent: we traverse the nodes that are only one or two steps away. Anchor-PROMPT is an extension of PROMPT that compares the graph structure on a larger scale. It takes as input a set of anchors-pairs of related terms defined by the user or automatically identified by lexical matching. Anchor-PROMPT treats an ontology as a graph with classes as nodes and slots as links. The algorithm analyzes the paths in the subgraph limited by the anchors and determines which classes frequently appear in similar positions on similar paths. These classes are likely to represent semantically similar concepts.

In terms of *user support*, the PROMPT tool has the following features:

- Setting the preferred ontology. It often happens, that the source ontologies are not equally important or stable, and that the user would like to resolve all the conflicts in favor of one of the source ontologies. We allow the user to designate one of the ontologies as preferred. When there is a conflict between values, instead of presenting the conflict to the user for resolution, the system resolves the conflict automatically.

- Maintaining the user's focus. Assuming a user is merging two large ontologies and is currently working in one content area of the ontology, PROMPT maintains the users focus by rearranging its lists of suggestions and conflicts and presenting first the items that include frames related to the topics of the latest operations.

- Providing feedback to the user. For each of its suggestions, PROMPT presents a series of explanations, starting with the reason why it suggested the operation in the first place. If PROMPT then changes the operation placement in the suggestions list, it also explains why it moved the operation.

URL: http://protege.stanford.edu/

ODEMerge

ODEMerge is a tool to merge ontologies that is integrated in WebODE, the software platform to build ontologies developed by the Ontology Group at Technical Universitiy of Madrid (see page 4.3.1). Therefore it is a *client-server tool* that works in the Web. This tool is a partial software support for the methodology for merging ontologies elaborated by de Diego (de Diego, R. *Método de mezcla de catálogos electrónicos*. Final Year Project. Facultad de Informática de la Universidad Politcnica de Madrid. Spain. 2001). This *methodology* proposes the following steps: 1) transformation of formats of the ontologies to be merged; 2) evaluation of the ontologies; 3) merging of the ontologies; 4) evaluation of the result; and 5) transformation of the format of the resulting ontology to be adapted to the application where it will be used.

The methodology establishes in a very detailed way: what tasks we need to carry out when we have to merge two ontologies, when we have to do these tasks, who must perform each task, how he must carry it out, and what the products of each tasks are. For the evaluation and merging of ontologies, very detailed rules are proposed. The methodology is based on the experience merging e-commerce ontologies.

WebODE helps in steps (1), (2), (4) and (5) of the merging methodology, and ODEMerge carries out the merge of taxonomies of concepts in step (3). Besides, ODEMerge helps in the merging of attributes and relations, and it incorporates many of the rules identified in the methodology. ODEMerge uses the following *inputs*: the source ontology 1 to be merged; the source ontology 2 to be merged; the *table of synonyms*, which contains the synonymy relationships of the terms of the ontology 1 with the terms of the ontology 2; the *table of hyperonyms*, which contains the hyperonymy relationships of the terms of the ontology 1 with the terms of the ontology 2.

ODEMerge processes the ontologies together with the information of the tables of synonymy and hyperonymy, and it generates a new ontology, which is the merge of the ontology 1 and the ontology 2. That is, the tool compares the ontology 1 with the ontology 2 considering the tables of synonymy and hyperonymy, and it merges these ontologies. An important characteristic of ODEMerge is that it can be used to merge ontologies in so many ontology implementation languages as the ones that WebODE processes, since WebODE is the host platform of ODEMerge. The WebODE import module allows importing ontologies written in XML, RDF(S) or CARIN, and allows exporting into XML, RDF(S), OIL, DAML+OIL, CARIN, FLogic, Prolog, Java and HTML.

URL: http://delicias.dia.fi.upm.es/webODE/

FCA-Merge

FCA-Merge is a *method* for merging ontologies, which follows a bottom-up approach offering a global structural description of the merging process. For the source ontologies, it extracts instances from a given set of domain-specific text documents by applying natural language processing techniques. Based on the extracted instances mathematically founded techniques are applied to derive a lattice of concepts as a structural result of FCA-Merge. The result obtained is explored and transformed into the merged ontology by the ontology engineer.

This method is based on application-specific instances of the two given ontologies O_1 and O_2 that are to be merged. The overall process of merging two ontologies consists of three steps, namely (i) instance extraction and computing of two formal contexts $K1$ and $K2$, (ii) the FCAMerge core algorithm that derives a common context and computes a concept lattice, and (iii) the interactive generation of the final merged ontology based on the concept lattice.

The extraction of instances from text documents avoids the problem that in most applications there are no objects which are simultaneously instances of the source ontologies, and which could be used as a basis for identifying similar concepts.

This method takes the two ontologies and a set D of natural language documents as input data. The documents have to be relevant to both ontologies, so that the documents are described by the concepts contained in the ontology. The documents may be taken from the target application, which requires the final merged ontology. From the documents in D, instances are extracted. This automatic knowledge acquisition step returns a formal context for each ontology indicating which ontology concepts appear in which documents.

The extraction of the instances from documents is necessary because there are usually no instances which are already classified by both ontologies. However, if this situation is given, one can skip the first step and use the classification of the instances directly as input for the two formal contexts.

The second step of the ontology merging approach comprises the FCA-Merge core algorithm. The core algorithm merges the two contexts and computes a concept lattice from the merged context using FCA techniques. More precisely, it computes a pruned concept lattice which has the same degree of detail as the two source ontologies.

Instance extraction and the FCA-Merge core algorithm are fully auto-

matic. The final step of deriving the merged ontology from the concept lattice requires human interaction. Based on the pruned concept lattice and the sets of relation names $R1$ and $R2$, the ontology engineer creates the concepts and relations of the target ontology.

For obtaining good results, a few assumptions have to be met by the input data: firstly, the documents have to be relevant to each of the source ontologies. A document from which no instance is extracted for each source ontology can be neglected for the task. Secondly, the documents have to cover all concepts from the source ontologies. Concepts that are not covered have to be treated manually after the merging procedure (or the set of documents has to be expanded). And last but not least, the documents must separate the concepts well enough. If two concepts that are considered as different always appear in the same documents, FCA-Merge will map them to the same concept in the target ontology (unless this decision is overruled by the knowledge engineer). When this situation appears too often, the knowledge engineer might want to add more documents that further separate the concepts.

Reference: B. Ganter, R. Wille: *Formal Concept Analysis: Mathematical foundations*. Springer, Berlin-Heidelberg 1999

Comments

The first thing we can notice is the higher level of *heterogeneity* of these tools with respect to the development tools. In fact the differences among the tools presented are more evident than in the previous case.

Merging tools are different in the *output* they can provide, in the *input* they need, in the *capabilities* they have during the merging process, in the *intervention* of the designer, besides the normal difference related to architecture and methodology.

Now we try to present some comments and observations to better understand better the characteristic of each tool.

We can start observing that PROMPT and ODEMerge are integrated in Protégé2000 and WebODE, respectively (see the previous sec.). Chimaera, instead, is based on a web-browser environment.

All the tools are able to integrate ontologies expressed in different *languages* (XML, RDF(S), OIL, etc.).

An important issue for merging tools is the *input* they need. Some tools deal only with class hierarchies of the sources and are agnostic in their merging algorithms about slots or instances (e.g. Chimaera). Other tools use not only classes but also slots and value restrictions in their analysis (e.g. PROMPT). Another set of tools require not only that instances are present,

but also that source ontologies share a set of instances (e.g. FCA-Merge). Besides the two ontologies to merge, ODEMerge eventually requires the table of synonyms and the table of hyperonyms in order to conduct the process.

Also regarding the *interactivity* with the designer there are different approaches. FCA-Merge works independently and produce suggestions to the user at the end, allowing the user to analyze the suggestions. PROMPT and Chimaera rely heavily on interaction with the designer and base their analysis not only on the source ontologies themselves but also on the merging or alignment steps that the user performs. ODEMerge performs a completely automatic process. During the integration process Chimaera and PROMPT present the user with possible alternatives with a graphical interface which allows the user to visually compare the source ontologies visually and accept or reject the results of the tools.

All the tools presented support the *conflict detection* and in particular they are able to detect name and structure conflicts. Furthermore all tools permit merging concepts, taxonomies, relations and instances.

With respect to the tool experience, PROMPT, Chimaera and ODE-Merge have been used with many different ontologies and different domains. Nevertheless PROMPT is one of the most frequently quoted tools in international papers. Studies to evaluate tools are frequent, one of them is [32] where PROMPT is tested following a defined framework for evaluating the quality of its suggestions.

4.3.3 Conclusion

After the presentation of this survey of development and integration tools, we can write some conclusions.

A first conclusion concerns *development tools*: they are partially out of our interest because their main objective is to help users in building ontologies, providing useful features for reducing the time necessary to build large ontologies.

Integration and merging tools, on the other side, are limited because they do not face all the problems related to dynamics. In particular it is important to notice that the main objective of integration tools is not to solve the dynamics issue but simply to help the designer during the integration process. For example none of the presented tools exploits the presence of previous versions to perform integration but they execute a process from scratch. With these observations we do not mean to doubt of the effectiveness and efficiency of the tools features, but we want to insert these tools' in the right context

with respect to our study.

In section 3.4 we described the complex operations that the designer has to do to keep the ontology updated, aligned and consistent after the creation. Moreover we have presented two of the main approaches to dynamics' problem: *evolution and versioning*. We can say that integration tools can not be discussed with respect to those approaches because the purpose of the tools is not to provide an environment for managing the dynamics' aspects of ontologies. In other words the tools are not able to support the complex modifications that ontologies need in order to remain consistent, and they are not able to exploit or manage multiple versions of ontologies or the previous results for preventing new integration process from scratch.

We can also argue that, with respect to ontology editor requirements (see sec. 4.2), *integration tools cannot be classified as ontology editors*. The main reason for this assumption is that ontology editors are inserted in a scenario where there are consistent ontologies, different changes are applied to ontologies and we want to keep them consistent. The integration tools, as we have seen, do not have the characteristics to face such a situation; none of the tool has the capability to manage changes, and in particular they do not have the notions of applying changes to a consistent ontology.

The study of development and integration tools has been useful for us to understand the state-of-the-art proposals and to evaluate the MOMIS system against the most important tools available. In particular, we can consider SI-Designer, the module of MOMIS for the integration process, as an integration tool. In this case we can say that SI-Designer is a good "integration tool" because, as we have shown before, it has similar characteristics to the other tools: software platform (Java), architecture (three layer architecture), import and export features (by means of wrappers from virtually any format and to XML and ODL_{I^3}), annotation of terms (with the WordNet lexical system), an internal inference engine (ODB-Tools), affinity and clustering technics and source navigation features.

Concerning the ontology editor and the possible approaches to dynamics, we can say that it is difficult to insert MOMIS in this context because the principal objective of MOMIS is to integrate sources in order to query the created global view using ontologies to execute the integration process. The study performed was motivated by the serious intention to face the dynamics aspects with the MOMIS system. In fact, the current version of MOMIS does not deal with the insertion of new sources and the modification or

the deletion of the previously integrated ones. Studying the state-of-the-art approaches we have understood the present limits and weak points of the MOMIS system and moreover we have acquired important knowledge in order to face the dynamics' issue. In the following chapter we will thoroughly explain the problems related to the management of dynamics aspects and we will propose a possible solution to the problem.

Chapter 5

Integration of new sources in the MOMIS system

As we have seen before, the dynamics' issue is a crucial aspect in several research areas such as Information Integration, Knowledge Representation, Ontology Management, Artificial Intelligence and Semantic Web.

In this chapter we will present the problems that arise when a new source (or more than one) has to be integrated in the MOMIS system. In fact, MOMIS is able to integrate sources by scratch. We will illustrate the current limits of the system, the problems that arise when we want to manage the dynamics, and possible approaches to solve them.

5.1 The static environment of MOMIS

In the previous chapters we have analyzed the new scenarios that the Semantic Web is creating all over the world. In particular we have seen how many research groups are working on projects related to the "new" Web, to the ontologies and their management, to the *dynamics*. Specially this last issue is becoming essential today.

Everything, in fact, evolves and changes rapidly: web pages, web applications, data of any type, ontologies, languages, models, methodology, etc. These elements are continuously modified to accomplish new requirements or to keep them consistent. So if someone wants to keep in contact with this new scenario, he has to face and solve the dynamics problem.

The MOMIS system is actually able to integrate heterogeneous sources in a semi-automatic way (see sec. **??** and 4.1 for details) but MOMIS was designed to solve the integration problem in a *static* environment. The MOMIS integration process, in fact, aims at building an integrated view of all the

sources involved. This view is also called ***Global Virtual View*** (or short **GVV**) because it allows the users to have a *Global View* on the integrated domain, but it is *Virtual* because it is not materialized. In other words, the *GVV* allows the users to look at the domain as if it is a unique source, leaving all the details for the management of the underlying sources to the system.

The present methodology of the MOMIS system is optimized to achieve the best results in the case in which the candidate sources to be integrated are all available at the beginning of the integration process. Besides, if the sources come from the same domain, that means they describe the same entities (such as persons, products, diseases, etc.) in different ways, the results will be probably better. The skill and the experience of the designer that follows and supervises the integration process are a key aspect to obtain good results.

The process is semi-automatic and therefore a designer or a domain expert has to interact with the system. Even if the SI-Designer (the integration tool of the MOMIS system) helps the user during every phase, the integration process is not trivial and it heavily depends on the dimension of the candidate sources. Similar observations are valid if we consider the computational effort to build the final view.

Before going on, it is better to explain what we exactly mean with "***dynamics management***" in the MOMIS system. ***We can define the management of the dynamics as the capability to face the request of inserting new sources and updating or deleting integrated sources.***

This definition raises several considerations and comments. The requests are normal and obvious in the scenario we have depicted before, but finding a solution to them is not easy and the following issues are of fundamental importance during the discussion of possible solutions.

1. ***Consistency.*** When the integration process is finished and the GVV has been generated, the system is in a *consistent state*. Any modification can lead to an inconsistent state of the system. The system has to manage this situation and reconcile the schema to a consistent state after each intervention.

2. ***Dependency.*** Users and applications rely on the GVV, that is users pose queries on the schema, and applications use the schema to operate. If the schema changes, unpredictable *side effects* on the queries' results and on the applications output may appear. Moreover some

optimizations may have been calculated relying on the present schema; recalculations are necessary if changes intervene.

In the second chapter we presented two approaches to the dynamics problem: Evolution approach (sec. 3.4.1) and Versioning approach (sec. 3.4.2).

The *evolution approach* is the most interesting attempt to formalize and to explain the whole complexity of the problem. This approach considers every implication that each change can have on the system. It can be useful in our context to help us to understand the possible side effects that each change can induce. But this study refers to an environment which is different from ours; the MOMIS system, in fact, is a source integration system and not an ontology management tool. So it is clear that we cannot follow all the requirements that the study proposes.

The *versioning approach* is surely of less interest for us. The proposal is to manage the changing scenario with versions of ontologies. In the MOMIS system is impossible to follow such a solution because it was designed for the creation of an integrated view of the interested domain.

As we have seen the problem is very complex. **In the following we will study only the case of the integration of new sources**. We will present the problems related to the introduction of new sources, explaining our choices and suggesting some possible solutions.

5.2 The problems faced

In this section we try to give more details about the problems related to the integration of new sources in the GVV.

We start analyzing the request: new sources have to be integrated in the schema. As we said before, the sources integrated usually come from a common domain (university, business, enterprise, etc.) and so they have some level of similarity. If the sources to integrate belong to the same domain of the GVV, there is higher probability to obtain a good result with few changes. Otherwise, if the sources are completely out of the domain, the final GVV will be probably quite different from the pre-existing one.

Now, in order to draw some guidelines for the discussion of the problem, we list a series of questions we will try to find an answer to.

- Is the Global Virtual View a good representation and a good synthesis of the knowledge of the integrated local sources?
- How can we save the knowledge hidden behind the GVV?

- Is it useful to save that knowledge?

- How can we use the knowledge of the local sources during the integration of new sources?

The answer to the first question is fundamental, because, if we consider the GVV as a good synthesis of the integrated sources, we have to compare the new sources with the GVV itself. On the other side, if we consider also the connections and the knowledge related to the local sources as important, the integration process has to take into account this assumption.

The GVV, as we have seen in the description of its construction, represents an integrated view of the local sources, but it saves also all the information that link each global entity to the corresponding local one (by means of the *mapping tables*). Moreover, the MOMIS system has an important source of knowledge: the *Common Thesaurus*. Therefore, the debate concerns the possibility to exploit the "local information" we have in the system to conduct the new integration process.

In case we decided to take advantage also of the information related to the old local sources, we have to study the best way to exploit this during the process.

A further aspect of the problem is the complexity of the present MOMIS process. In fact, the integration process is expensive for the designer, who has to interact with the system, and for the system, which has to support the designer during the process. These claims are important in order to decide the strategy to be followed to solve the problem. *One of the possible and probably the most simple solution is to* ***restart the process from scratch*** *involving old and new local sources.*

We did not consider this solution as good solution because it did not exploit the work of the previous integration process and thus, in the following we will not consider it anymore.

In order to improve our study, we observe that in the present integration process all the sources to be integrated equally concur to the process. Therefore, if we suppose that the domain described by a MOMIS GVV is quite static and new sources to be integrated belong to that domain, we may assume that a new source brings less semantics than the GVV. Therefore, ***we assume an integration process of the new source that starts from the GVV obtained and tries to integrate this new source in the GVV.***

5.3 A first attempt

We are now going to present the heuristic approach which was initially used, trying to exploit the previous assumptions related to dynamics. This solution has been then abandoned, but it has been useful to understand some defects of the present MOMIS integration approach.

Let us analyze the final results of the integration process. The Global Virtual View is mainly composed of global classes, each one with its mapping table. We observe that global classes represent a well defined integrated view of the local sources, but they are not related to each other. The relationships saved in the Common Thesaurus are referred to local sources and they are used only to determine clusters.

The idea was to better exploit the knowledge contained in the Common Thesaurus. We thought that during the integration of new sources against the GVV, it could be useful to have relationships among global classes. In some ways it was an attempt to preserve some of the knowledge saved in the Common Thesaurus.

We have to transform the Common Thesaurus in order to exploit it during the integration of new sources (see figure 5.1).
Before going on, we recall here the structure of a CT (Common Thesaurus)

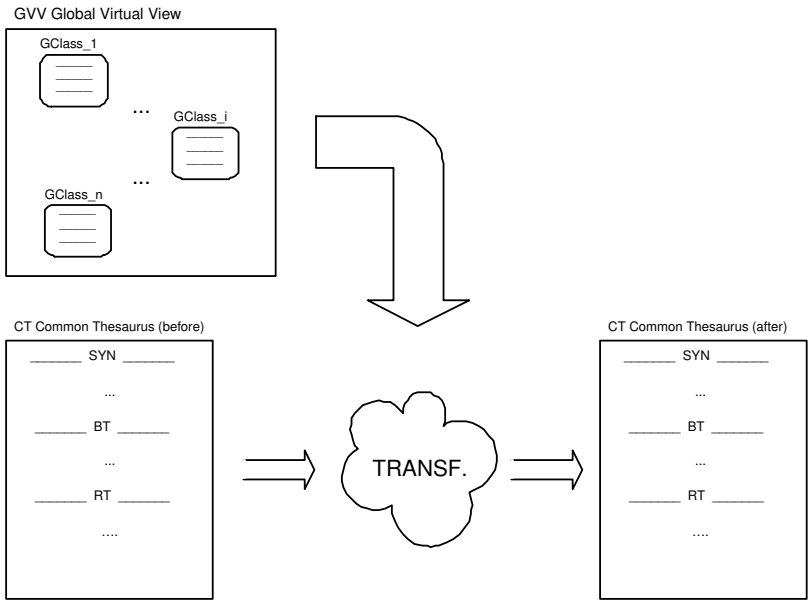

Figure 5.1: The Common Thesaurus transformation.

relationships:
- Name path of the left side of the relationship;
- Type of relationship: SYN, NT, BT, RT;
- Name path of the right side of the relationship.

The name path is expressed in "dot notation":
- `NameLocalSource.NameLocalClass` or
- `NameLocalSource.NameLocalClass.NameLocalAttribute`

The relationships can involve different elements:
- Local Class ↔ Local Class or
- Local Class ↔ Local Attribute or
- Local Attribute ↔ Local Class or
- Local Attribute ↔ Local Attribute.

For example we can have:
- `UNI.Room RT UNI.Section`
- `CS.Location BT CS.Office.address`
- `UNI.School_member.faculty RT CS.CS_Person`
- `UNI.Section.room_code SYN UNI.Room.room_code`

At the end of the integration process each local entity is mapped in a global one: local classes are mapped in clusters (i.e. global classes) and local attributes, by means of the mapping tables, are mapped into global attributes. Relying on this knowledge, *we propose to update CT relationships in order to have a new set of relationships among global entities (classes and attributes)*. The rules to elaborate the CT relationships are listed below and are divided into three groups:

1. **Global Class ↔ Global Class**
   The relationship involves two local classes that are mapped in two different global classes. The sources of the local classes are of no importance. The type of relation does not change. The path name changes with respect to the global class, that is, the source name becomes the name of the global schema defined by the designer at the beginning of the integration process, while the local class name becomes the name of the global class in which it is mapped.

   `UNI.Room RT UNI.Section` (before the transf.)
   `GVV`[1]`.Room RT GVV.Course` (after)

[1]GVV is considered the name of the Global Schema chosen by the designer.

`UNI.Research_Staff RT CS.Course` (before)
`GVV.Person RT GVV.Course` (after)

The relationships are taken from the complete example presented in appendix A.

2. **Global Class ↔ Global Attribute**
The relationship involves a local class and a local attribute that are mapped in two different global entities. The global entities can belong or not to the same global class. The source name becomes the name of the global schema, while the local class name becomes the name of the global class in which it is mapped, and the local attribute takes the name of the corresponding global attribute.

`CS.CS_Person BT UNI.Article.author` (before the transf.)
`GVV.Person BT GVV.Publication.authors` (after)

The relationship involves two different global classes (`Location` and `Office`).

`CS.Professor RT UNI.School_Member.faculty` (before)
`GVV.Person RT GVV.Person.faculty` (after)

`UNI.Department SYN UNI.Department.dept_name` (before)
`GVV.Department SYN GVV.Department.dept_name` (after)

The relationships involve the same global classes (`Person` and `Department`).

3. **Global Attribute ↔ Global Attribute**

The relationship involves two local attributes that are mapped in two different global attributes. Like before, the source name becomes the name of the global schema, while the local class name becomes the name of the global class in which it is mapped and the local attribute takes the name of the corresponding global attribute.

`UNI.Section.room_code SYN UNI.Room.room_code` (before the transf.)
`GVV.Course.room_code SYN GVV.Room.code` (after)

The relationship involves two different global classes (`Course` and `Room`) but it can also involve the same global class.

The updated relationships are then saved in a data structure similar to the one used with the CT relationships; they will be exploited during the integration of new sources. This transformation enriches the global schema, in fact, now we can know the relationships among the global entities (classes and attributes).

At this point a further fundamental assumption is necessary. In order to define the new integration process we consider that: *the global schema, which was previously built, with the new relationships just created, is a good synthesis of the integrated local sources. This claim allows us to compare the new candidate sources with this enriched schema.*

As a consequence, we suggest a new integration process that exploits the present architecture of the MOMIS system bringing only small changes. The idea is to transform the global schema with the aims of simulating a new source. Since the global schema does not have a "*real*" structure from which we can extract relationships, we take advantage of the relationships we have obtained before. In this way we can use the normal integration process of MOMIS but this time there are:

- the Global Virtual View;

- the new sources.

The phases of the previous process remain the same except the extraction of intra and inter schema relationships that is not performed on the Global Virtual View.

This solution exploits the feature of the MOMIS system to export the Global Schema as a XML file and to load a schema from an XML file. In this way we force the system to accept the GVV as a new source; in addition we have to modify the process in order to load also the updated CT relationships (referred to the GVV) from the XML file, instead of calculating them.

We wrote the code to accomplish these directions. We create a new tab in the MOMIS system that guides the designer in the creation of the XML file. This file has to simulate a new source in the integration of further sources. Thanks to the tabs, the user can:

- update the Common Thesaurus relationships following the rules described above;

- create the XML file that simulates a source.

We tested this process and we obtained good results. Tests were carried out on few examples and therefore we do not have significant data to evaluate this process completely.

This first heuristic attempt has been very useful because we have understood some important defects of the present process. In particular we have observed that the result of the integration process, the Global Virtual View, has to be modified to become a good synthesis of the local sources. We tried to solve these problems with the solution we have just presented, but during the study we understood that the problems were concentrated around the GVV. We summarize here two main defects we found:

- global classes are separated, there are no relationships among them;

- global terms (names of classes and attributes) are not lexically annotated.

5.4 The proposed solution

Exploiting the previous study, we present here a new solution.

The solution we propose is based on the assumption that the present integration process is effective and gives good results.

This assumption is coherent with what we stated before about our intention to save the present MOMIS approach and its results as much as possible. Therefore, working with this in mind, we identified two main problems to be solved:

Global Schema annotation. The global schema has to be semantically enriched if we want to improve the expressive power of the schema. This phase is fundamental to prepare the GVV for the integration with new sources. Moreover the annotation step allows us to save as much knowledge as possible in the GVV to be exploited in next integration sessions.

New integration process. The present integration process must be changed in order to accept new sources and the previous GVV semantically enriched as input.

The following sections will explain the details of these two important issues.

5.4.1 Global Schema annotation

The *Global Schema annotation* extends the present version of the MOMIS framework. The integration process builds global classes (see sec. 4.1.2 for details) but it does not annotate them. As said before, the annotation phase will enrich the global schema and prepare it to be used for new integration processes.

Before going on we specify what we mean with annotation: ***the annotation process wants to assign a name and a meaning (at least one) to global classes and attributes***.

We point out that the local terms are annotated during the present MOMIS process. Now we explain what is to be done to annotate the global schema presenting all the possible cases.

In the following we will use this notation:

- Global Class $gc_k = \{lc_1, \ldots, lc_n\}$ where $n \geq 1$, is composed of a set of Local Classes, lc_i, belonging to the Cluster Cl_k associated to gc_k
- Local Class $lc_i =< lcName_i, \{lcM_{i1}, \ldots, lcM_{im}\} >$ where $m \geq 0$, is composed of a name, $lcName_i$, identifying unambiguously the local class and a set of meanings lcM_{ij}

Global Classes' annotation

The operation of semantically annotating a Global Class consists of assigning a representative name (unique in the schema) and a set of meanings to it. We recall that a Global Class represents a set of local classes related to each other by a high affinity level. The name of the Global Class is automatically set by the system and therefore it is not significant; it has the structure $Global_i$ where i is the global class counter. Afterwards the designer has to set a significant name for each Global Class. What we want it is to eventually have an annotated Global Class:

$$gc_k =< gcName_k, \{gcM_{k1}, \ldots, gcM_{kp}\} > \text{where } p \geq 0$$

Concerning *the Global Class name* we consider it as a label; its principal role is, in fact, to help the designer to identify the Global Class and its contents, so in some cases the selected name could not match with a word form of WordNet (e.g. `School_Member`), or the chosen name may not be represented by the meanings of the global class.

Concerning *the global class meaning*, we assume that each meaning has a unique identifier within WordNet, let us say M. In order to associate a word form F to the right meaning M we use $F\#counter$, that is we use the element of the lexical matrix of WordNet:

Word Form Meaning	F	...
M	$F\#counter$	
...		

In this way a meaning can be identified also with $F\#counter$; furthermore each meaning can have more than one of these identifiers. In our examples we will use this type of notation.

The annotation process can be achieved semi-automatically following some rules; of course, the designer may always refine what the system suggests. The possible cases are:

a) **Only one Local Class belongs to the Global Class** ($n = 1$). In this case, both the name and the meaning of the global class are the same of the Local Class. If no meaning was provided before by the designer for the Local Class, the designer is asked here to supply the missing information.

The clustering result is:

$\texttt{Global}_3$ = {CS.Office}

where

CS.Office = <office,{office#1}>

thus we automatically obtain:

$\texttt{Global}_3$ = <office,{office#1}>

office#1 = place of business where professional or clerical duties are performed

b) **More than one Local Class belongs to the Global Class** ($n > 1$).
We examine the following cases:

1. *There is a synonymy relationships among all the Local Class names.*
   In this case, the system has to assign the union of the local classes meanings to the meaning of the Global Class, and for the name, the designer has to choose the more significant term.
   Notice that, if a synonymy relationship $lcName_i$ `SYN` $lcName_j$ was generated by the lexical analysis, then the corresponding meanings are the same.

   The clustering result is:

   $\texttt{Global}_0$ = {CS.Course, UNI.Section}

   where

   CS.Course = <course,{course#1}>
   UNI.Section = <section,{course#1}>

   and with the following relation:

   CS.Course SYN UNI.Section

   following the rules, the global class is:

   $\texttt{Global}_0$ = <course,{course#1}>
   or
   $\texttt{Global}_0$ = <section,{course#1}>

   course#1 = education imparted in a series of lessons or class meetings

2. *All the class names are connected only by narrow (broader) term relationships.* The name and the meaning of the Global Class are derived from the broadest local class of the cluster.

   The clustering result is:

   $\texttt{Global}_5$ = {CS.Essay, CS.Publication, UNI.Article}

where

CS.Essay = <essay,{essay#1}>
CS.Publication = <publication,{publication#2}>
UNI.Article = <article,{article#1}>

and with the following relations:

UNI.Article NT CS.Publication
CS.Essay NT CS.Publication

following the rules:

$Global_5$ = <publication,{essay#1,publication#2,article#1}>

essay#1 = an analytic or interpretive literary composition
publication#2 = a copy of a printed work offered for distribution
article#1 = nonfictional prose forming an independent part of a publication

3. *All the class names are connected by synonymy and narrow (broader) term relationships.* If it is possible to calculate the transitive closure of the meanings, then the system suggests the broadest meaning. Otherwise, see next case.
The clustering result is:

$Global_4$ = { CS.CS_Person, CS.Professor, UNI.School_Member, UNI.Research_Staff, CS.Student}

where

CS.CS_Person = <person,{person#1}>
CS.Professor = <professor,{professor#1}>
UNI.School_Member = <student,{student#1}>
UNI.Research_Staff = <researcher,{professor#1}>
CS.Student = <student,{student#1}>

and with the following relations:

CS.Professor NT CS.CS_Person
UNI.School_Member NT CS.CS_Person

```
UNI.Research_Staff SYN CS.Professor
UNI.Research_Staff NT CS.CS_Person
UNI.School_Member SYN CS.Student
CS.Student NT CS.CS_Person
```

following the rules:

$\texttt{Global}_4$ = <person,{person#1,professor#1,student#1}>

person#1 = a human being
professor#1 = someone who is a member of the faculty at a college or university
student#1 = a learner who is enrolled in an educational institution

4. *Other cases.* In every other case, the designer has to manually select the name and the meaning(s) of the Global Class.

Given a Global Class gc, a set of Local Classes belonging to gc $(lc_1, \ldots, lc_n)$, the next table summarizes the described cases (Table 5.1).

Case	Local Class	Global Class
1	$lcName_i$ SYN $lcName_j$ for each i, j	$gcName = lcName_1 or lcName_2 or \ldots$ $gcM = \cup_i \{lcM_{i1}, \ldots, lcM_{im}\}$
2	$lcName_i$ BT $lcName_j$ for each $i \neq j$	$gcName = lcName_i$ $gcM = \{lcM_{i1}, \ldots, lcM_{im}\}$
3	$lcName_1$ BT ($lcName_2$ SYN/BT $lcName_n$)	$gcName = lcName_i$ $gcM = \{lcM_{11}, \ldots, lcM_{1m}\}$
4		$gcName = lcName_1 or \ldots lcName_n$ $gcM \subseteq \cup_i \{lcM_{i1}, \ldots, lcM_{im}\}$

Table 5.1: Global Class annotation rules.

When gcM is different from the union of all the local classes meanings, some semantics (lexical relationships) is lost. To avoid this loss of semantics, we can consider all the meanings of the local classes belonging to a Global Class, i.e. we could have $gcM = \cup_i \{lcM_{i1}, \ldots, lcM_{im}\}$.

Global Attributes' annotation

In order to obtain as much as possible an automatic annotation we propose to provide to the designer the solution explained before. In this way, the designer must only refine the suggestions he does not like.

With respect to the complete reference example presented in Appendix "A" we can have the following situation:

Global Class **Global**$_4$ (**Person**)

	CS	CS	CS	UNI	UNI
	CS_Person	Professor	Student	Research_Staff	School_Member
belongs_to	null	belongs_to	null	null	null
dept_code	null	null	null	dept_code	null
e_mail	null	null	null	e_mail	null
faculty	null	null	null	null	faculty
name	first_name and last_name	first_name and last_name	first_name and last_name	name	name
rank	null	rank	rank	null	null
relation	null	null	null	relation	null
section_code	null	null	null	section_code	null
title	null	title	null	null	null
year	null	null	year	null	year

the local attributes have been annotated:

CS.Student.first_name = <first_name,{first name#1}>
CS.Student.last_name = <last_name,{last name#1}>
CS.Professor.name = <name,{name#1}>
CS.Professor.rank = <rank,{rank#2}>
CS.Student.rank = <rank,{rank#4}>
CS.Professor.title = <title,{title#6,tile#9}>

first name#1 = the name that precedes the surname
last name#1 = the name used to identify the members of a family
name#1 = a language unit by which a person or thing is known
rank#2 = relative status
rank#4 = position in a social hierarchy
title#6 = an identifying appellation signifying status or function
title#9 = an appellation signifying nobility

we also have these relations:

CS.CS_Person.first_name NT UNI.Research_Staff.name
CS.CS_Person.last_name NT CS.Research_Staff.name

```
CS.CS_Person.first_name NT UNI.School_Member.name
CS.CS_Person.last_name NT CS.School_Member.name
CS.Student.rank SYN CS.Professor.rank
```

At this point, we may distinguish three possible cases:

1. *The Global Attribute maps only one Local Attribute.* In this case, the Global Attribute has the same name and the same meaning of the mapped Local Attribute.

   $\texttt{Global}_4\texttt{.title = <title,\{title\#6,tile\#9\}>}$

2. *The Global Attribute maps more than one Local Attribute (only with simple correspondence).* The name of the Global Attribute is chosen according with the rules defined in the present MOMIS framework. Regarding the meaning choice, we distinguish two possible solutions, similarly to the Global Class case. In fact, the Global Attribute may be assigned the meaning corresponding to the Local Attribute, to which the name refers. In this case, some further relationships between this attribute and a hypothetic new source can not be extracted (we lose some of the meanings mapped by the Global Attribute). The second approach assign the meanings as the union of the meanings of the mapped Local Attributes.

   $\texttt{Global}_4\texttt{.rank = <rank,\{rank\#2,rank\#4\}>}$

3. *The Global Attribute maps more than one Local Attribute (correspondence defined by functional mappings).* The rule is the same of point 2. In particular, according to the mapping function, we may develop some ways in order to ignore some meanings (transitive closure calculation, ...), or to prefer other ones.

   $\texttt{Global}_4\texttt{.name = <name,\{first name\#1,last name\#1,name\#1\}>}$

In all other cases we consider all the mappings, and the designer will select a significant name.

5.4.2 The new integration process

As we have just seen, the annotation phase enriches the semantics of the Schema adding significant names and meanings to global terms. The Global Schema is now a satisfactory synthesis of the local sources and it is ready to be used as a component of the new integration process.

The new integration process involves: the GVV, previously enriched, and the new source schemas extracted by wrappers.

In other words, the global classes of the GVV are considered as local classes and they are integrated with the local classes of the new sources. As mentioned, this methodology aims at obtaining a new GVV reducing relevant changes. This aspect guarantees a minor impact for the applications based on the previous view. In fact, other modules or applications could have been optimized for the old GVV, therefore that view has to be saved from useless changes. Figure 5.2 schematically represents the new integration process and in particular the components that are involved.

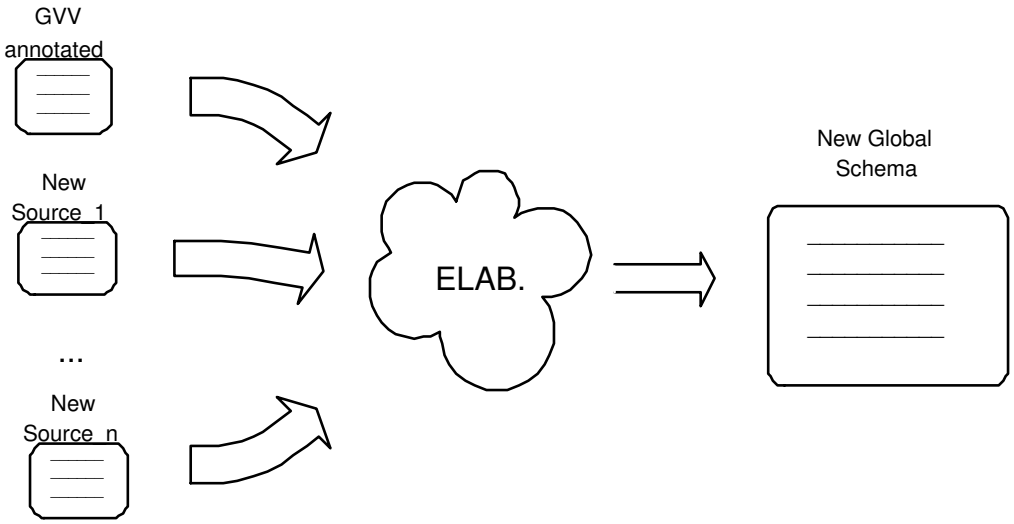

Figure 5.2: The new integration process.

Before going on, we define the meaning of some symbols we will use later.

gcOld is a Global Class of the old integrated schema. It is composed of a name (*gcOldName*) and a set of global attributes ($gcOldAtt_j$).

lcNew is a Local Class of the new source. It is composed of a name (*lcNewName*) and a set of local attributes ($lcNewAtt_k$).

$gcNew$ is a Global Class of the new integrated schema. It is composed of a name ($gcNewName$) and a set of global attribute ($gcNewAtt_i$).

In accordance with the usual integration methodology (see sec. 4.1) we summarize the steps to be done in order to integrate the new sources strengthening the principal differences:

1. the wrappers extract the schemata from the new candidate sources and translate them into the MOMIS common language, ODL_{I^3};
2. the Common Thesaurus of the involved sources is built up and contains:
   - schema-derived relationships extracted from the analysis of the new sources (*old global classes do not add significant relationships*);
   - inter-schema lexicon-derived relationships (new sources will be annotated by the designer and old global classes have to be semantically enriched according to the rules shown in 5.4.1);
   - user supplied relationships;
   - inferred relationships;
3. Clusters generation is made by using the usual methodology;
4. Global Classes' and mapping tables' refinement is performed;
5. annotation of the Global Schema as described before is performed.

Let us express some important considerations about this new approach.

The integration process is based fundamentally on lexical semantics. In fact, as we have seen also presenting the first attempt, we cannot exploit the relationships contained in the Common Thesaurus because these relationships involve local sources referring to the previous integration session. Moreover, we cannot extract any relationship from the Global Schema because Global Classes are flat, that is, they do not have any type of structure from which relationships can be inferred. Therefore we have to define new strategies to find out relationships among Global Classes if we want to effectively integrate the GVV with new sources. The solution we propose is the annotation process of global terms with respect to the lexical database WordNet.

If we analyze this aspect more deeply, we can say that it is possible that new sources have some lexical relationships with a concept not annotated in a Global Class.

For example, let us consider the case in Figure 5.3; if the annotated meaning within the cluster composed of $\{person, professor\}$ is only the broadest (i.e. *person*), the relationship with the new class $faculty$ will be lost. In fact, the concept belonging to the new source is a holonym of a concept associated to a local class, and this relationship is not transitive. With re-

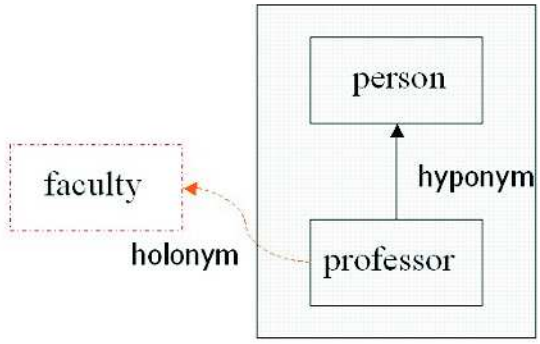

Figure 5.3: Lexical problems.

spect to a previous hypothesis for considering the union of the local classes' meanings in the annotation of a global class, we can illustrate some possible side effects. In the previous example, if the global class is annotated with both the meanings of *person* and $professor$, the holonym relationship between $faculty$ and $professor$ will be promoted to the global class. The result is that all elements belonging to this global class will have a holonym relationship with $faculty$ and in this specific case, will be part of a $faculty$. This partially wrong result will be corrected within the Common Thesaurus, where further relationships generated by other terms of involved sources are present.

If we consider the Global Class and mapping table refinement, we can say that the objective of this phase is to provide mapping rules among *Global Classes* and new, or old, *Local Classes*. In order to achieve this result, we substitute *old Global Glasses* with the respective *Local Classes* preserving all the previous mappings. In this way, new Global Classes, representing old Local Classes and new Local Classes, are built. Following this approach, we can represent a Global Class as a set of old Global Glasses and new Local Classes by using the following notation:

$$gcNew = \{gcOld_1, \ldots, gcOld_p, lcNew_1, \ldots, lcNew_n\}$$

the result of the rewriting step will be:

$gcNew = \{lcOld_{11}, \ldots, lcOld_{1z}, \ldots, lcOld_{p1}, \ldots, lcOld_{pn}, lcNew_1, \ldots, lcNew_n\}$.

For example:

```
gcNew = {GVV.Room, Math.ClassRoom}
```

that becomes:

```
gcNew = {CS.ClassRoom, UNI.Room, Math.ClassRoom}
```

The following are the possible cases that can arise in the creation of global classes (see also Appendix "A"):

Case A. A new global class $gcNew$ is composed of only one old global class ($gcOld$), and one or more than one new local classes ($lcNew_i$):

$$gcNew = \{gcOld, lcNew_1, \ldots, lcNew_i, \ldots, lcNew_n\}$$

```
gcNew = {GVV.Person,Math.Person,Math.Researcher,
         Math.Student,Math.Teacher}
```

Regarding this case we can observe that new sources' classes can be "integrated" by the way of the old GVV, thus, without effects on the applications based on that previous schema.

The new global class ($gcNew$) may have new global attributes generated from the semantic contribution of new local classes. In this case, the new global class has a name ($gcNewName$), a set of global attributes derived from the old global class ($gcOldAtt_i$) and a set of new global attributes ($gcNewAtt_j$).

New mapping rules define the connection between a global attribute and its corresponding local attribute(s). In this situation, global attributes belonging to the $gcOld$ ($gcOldAtt_i$) can map both local classes of the previous GVV and new local classes (see the column associated to $lcNew_1$, for example). New global attributes can only map new local classes (see the null cells in the table).

With respect to the meanings associated to each global attribute we have that:

- the meaning(s) of old global attributes have to be enriched with the semantics of the new local classes possibly mapped by this attribute;

<table>
<tr><td></td><td>$lcOld_1$</td><td>...</td><td>$lcOld_k$</td><td>$lcNew_1$</td><td>$lcNew_t$</td><td>$lcNew_n$</td></tr>
<tr><td>$gcOldAtt_1$</td><td colspan="3" rowspan="3">the same mappings as in $gcOld$</td><td colspan="3" rowspan="6">new mappings</td></tr>
<tr><td>...</td></tr>
<tr><td>$gcOldAtt_m$</td></tr>
<tr><td>$gcNewAtt_1$</td><td colspan="3" rowspan="3">null mappings</td></tr>
<tr><td>...</td></tr>
<tr><td>$gcNewAtt_p$</td></tr>
</table>

Table 5.2: New mapping table example.

- the meanings(s) of new global attributes have to be set according to the rules defined before (sec. 5.4.1).

Case B. A global class of the new integrated schema is composed of only new local classes:

$$gcNew = \{lcNew_1, \ldots, lcNew_i, \ldots, lcNew_n\}$$
$$\texttt{gcNew = } \{\texttt{Math.Examination}\}$$

This situation describes the case where the schema is extended without interfering with the previous classes.
The new global class ($gcNew$) has a name ($gcNewName$) and contains a set of new global attributes ($gcNewAtt_i$).
The new global attributes map only new local classes.
The names and meanings of the global attributes are defined following the rules stated before (sec 5.4.1).

Case C. A global class of the new integrated schema is composed of more than one global class of the GVV and at least one local class of the new sources that we are integrating:

$$gcNew = \{gcOld_1, \ldots, gcOld_p, lcNew_1, \ldots, lcNew_i, \ldots, lcNew_n\}$$

In this case the previous GVV is modified; side effects can influence the applications based on the previous schema.
The new global class ($gcNew$) has a name ($gcNewName$) and a set of new global attributes ($gcNewAtt_i$).
If an old global attribute ($gcOldAtt_j$) is not "fused" with another global attribute belonging to a different old global class, no change to the previous mapping rules is necessary. In other words, for this global

attribute all the previous mappings, w.r.t. the previous *lcOld*, are valid. Otherwise we have to define new mappings.

It can be demonstrated that these are the three only possible cases. This claim remains true only if the same clustering parameters are used during the new integration process.

5.5 Necessary changes

The present framework of the MOMIS system is not able to accomplish the presented approach, so some changes are necessary. Now we try to analyze the main modifications that have to be designed and realized in the future.

According to the approach we have illustrated, two main parts of intervention are: I) global schema annotation and II) new sources integration. These phases are managed by SI-Designer component, therefore changes will interest its modules (see sec. 4.1).

I) Global Schema annotation.

First of all, the global class building and refinement process have to be changed in order to allow the designer to annotate global classes and attributes. The SI-Designer modules involved are:

- ARTEMIS: at present this module computes global clusters exploiting affinity indexes. New features have to be added to guide the designer after the creation of the global classes. In particular we need features in order to:
  - select a significant name for each global class, suggesting possible names to the designer according to the rules explained before in sec. 5.4.1;
  - assign the correct meanings to each global class suggesting possible meanings to the designer according to rules in sec. 5.4.1.
- TUNIM: at present this module computes global attributes and helps the designer during the refinement phase. New features have to be added to accomplish the new rules defined in sec. 5.4.1, that is to enrich global attributes with meanings derived from local attributes annotation. In particular we need features in order to:
  - select a significant name for each global attribute;

– assign the correct meanings to each global attribute suggesting possible meanings to the designer according to rules in sec. 5.4.1.

II) New sources' integration.

After the enrichment process we need to modify the integration process to be able to integrate new sources according to the proposal of sec. 5.4.2. Mainly we have to modify:

- SAM: at present this module allows the designer to import sources to be integrated. New features have to be added to import the previous GVV and to treat it as candidate source.
- SIM: at present this module extracts Common Thesaurus relationships from the local schemata and infers new relationships using ODB-Tools. Changes are necessary to exclude that the module treats the previous GVV as a normal schema.
- SLIM: this module allows the designer to annotate the local terms. We have to modify the module in order to exploit the global schema annotation.

5.6 Critical analysis of the proposed process

We have introduced a new integration process approach, now we present some considerations about advantages and drawbacks we have noticed during the study.

Benefits :

- The process does not restart from scratch and therefore the designer's effort and the computational effort are reduced.
- The process exploits the knowledge acquired during the previous integration sessions (especially annotation); the time for integrating is remarkably reduced.
- The previous schema is preserved as much as possible from useless changes and therefore the possible side effects on the based applications are reduced.
- The present methodology of MOMIS is preserved, only some additions and modification are necessary.

Drawbacks :

- We must annotate the GVV, before integrating new sources.
- The mistakes of the previous integration session can propagate to the new schema.
- The new GVV is based on the previous one, and then it could not represent all the local sources in the best way.
- Repeated sessions of integration can reduce the expressive power of the integrated schema.

Chapter 6

Semantic Web projects

All around the world several universities, organizations, research communities started projects related to *Semantic Web*. In this chapter we will present some projects and languages of particular interest for this thesis. To realize this overview we have chosen the most important works or the most cited papers in literature and we have tried to understand how the dynamics problem is faced and which are the most valuable proposals to solve it. This presentation is not complete and exhaustive; we have studied these projects in order to evaluate our approach to dynamics. Further information can be found following the links present in this chapter.

The **W3C consortium** has a particular group of people that are studying new languages (**OWL** *Web Ontology Language* for example) and new standards related to the new tools that begin to appear on the web. The European Union by means of the IST (Information Society Technologies) Programme supports different projects involved in the same area of interest; we will give information about **OntoWeb** and **On-To-Knowledge** that have done many studies and involve many researchers from academic and industrial area. Some important universities are also involved in the study of the problem. Karlsruhe University is working on a framework for the management of the dynamics of ontologies (**MAFRA** - **Kaon**), which is very interesting for its completeness. Finally we will present **SHOE**, a language for the Semantic Web proposed by J. Heflin and J. Hendler of the University of Maryland.

6.1 W3C Consortium

The **World Wide Web Consortium**[1] was created in October 1994 to lead the World Wide Web to its full potential by developing common protocols that promote its evolution and ensure its interoperability. W3C has around 500 Member organizations from all over the world and has earned international recognition for its contributions to the growth of the Web.

W3C Mission is to promote interoperability and to encourage an open forum for discussion and it commits to leading the technical evolution of the Web. W3C's long term goals for the Web are: 1. *Universal Access*, to make the Web accessible to all by promoting technologies that take into account the vast differences in culture, languages, education, ability, material resources, and physical limitations of users on all continents; 2. *Semantic Web*, to develop a software environment that permits each user to make the best use of the resources available on the Web; 3. *Web of Trust*, to guide the Web's development with careful consideration for the new legal, commercial, and social issues raised by this technology.
W3C concentrates its efforts on three main tasks:

- *Vision*: W3C promotes and develops its vision of the future of the World Wide Web. Contributions from several researchers and engineers enable W3C to identify the technical requirements that must be satisfied if the Web is to be a truly universal information space.

- *Design*: W3C designs Web technologies to realize this vision, taking into account existing technologies as well as those of the future.

- *Standardization*: W3C contributes to efforts to standardize Web technologies by producing specifications (called "Recommendations") that describe the building blocks of the Web. W3C makes these Recommendations (and other technical reports) freely available to all.

W3C Activities are generally organized into groups: *Working Groups* (for technical developments), *Interest Groups* (for more general work), and *Coordination Groups* (for communication among related groups). These groups produce: technical reports, open source software, and services (e.g., validation services). These groups also ensure coordination with other standards bodies and technical communities. There are currently over thirty W3C Working Groups.

[1] http://www.w3c.org/

One of them works in the Semantic Web area[2]. They rely their activities on the considerations that facilities to put machine-understandable data on the Web are becoming a high priority for many communities. The Web can reach its full potential only if it becomes a place where data can be shared and processed by automated tools as well as by people. For the Web to scale, tomorrow's programs must be able to share and process data even when these programs have been designed totally independently. ***The Semantic Web***, they state, ***is a vision***: *the idea of having data on the web defined and linked in a way that they can be used by machines not just for display purposes, but for automation, integration and reuse of data across various applications.*

6.1.1 OWL Web Ontology Language

In November 2001, the W3C started a **Web Ontology Working Group** to define a language. This group is committed to take DAML+OIL as its starting point and is developing a language called **OWL *Web Ontology Language***. The OWL is intended to provide a language that can be used to describe the classes and relations between them that are inherent in Web documents and applications. The OWL is being designed in order to provide a language that can be used for applications that need to understand the content of information instead of just understanding the human-readable presentation of content. OWL allows more machine readability of web content than XML, RDF, and RDF-S support by providing an additional vocabulary for term descriptions. The OWL language is a revision of the DAML+OIL web ontology language incorporating learnings from the design and application use of DAML+OIL. *OWL can be viewed as an extension of a restricted view of the RDF language. This implies that every OWL document is an RDF document, but not all RDF documents are OWL documents.*

OWL is a language for defining *Web ontologies* and their associated knowledge bases. In OWL, an *ontology* is a set of definitions of classes, properties, and constraints on the way those classes and properties can be employed. *An OWL ontology may include the following elements*:

- taxonomic relations between *classes*;
- *datatype properties*, descriptions of attributes of elements of classes;
- object properties, descriptions of relations between elements of classes;

[2]http://www.w3c.org/sw/

and, to an inferior degree:

- *instances* of classes;
- *instances* of properties.

Datatype properties and object properties are collectively the *properties* of a class.

A set of OWL assertions loaded into a reasoning system is called a knowledge base (KB). These assertions may include facts about individuals that are members of classes, as well as various derived facts, facts not literally present in the original textual representation of the ontology, but entailed (logically implied) by the semantics of OWL. These assertions may be based on a single ontology or multiple distributed ontologies that have been combined using OWL mechanisms.

OWL is a set of three, increasingly complex languages.

OWL Lite has been defined with the intention of creating a simple language that will satisfy users primarily needing a classification hierarchy and simple constraint features. For example, while it supports cardinality constraints, it only permits cardinality values of 0 or 1.

OWL DL includes the complete OWL vocabulary, interpreted under a number of simple constraints. Primary among these is type separation. Class identifiers cannot simultaneously be properties or individuals. Similarly, properties cannot be individuals. OWL DL is so named due to its correspondence with *description logics*.

OWL Full includes the complete OWL vocabulary, interpreted more broadly than in OWL DL, with the freedom provided by RDF. In OWL Full a class can be treated simultaneously as a collection of individuals (the class *extension*) and as an individual in its own right (the class *intension*). Another significant difference from OWL DL is that a `DatatypeProperty` can be marked as an `InverseFunctionalProperty`.

An OWL ontology is made up of several components, some of which are optional, and some of which may be repeated. Now a simple example of an OWL ontology is presented.

```
<rdf:RDF
    xmlns     ="http://www.example.org/wine#"
    xmlns:vin ="http://www.example.org/wine#"
    xmlns:food="http://www.example.org/food#"
```

```
    xmlns:owl ="http://www.w3.org/2002/07/owl#"
    xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:xsd ="http://www.w3.org/2000/10/XMLSchema#"
    xmlns:dte ="http://www.example.org/wine-dt#">
```

A typical OWL ontology begins with a *namespace declaration*. The first two declarations identify the namespace associated with this ontology. The first makes it the default namespace, stating that unprefixed elements and empty URI references refer to the current ontology. To state the beginnings of the ontology we use

```
<owl:Ontology rdf:about="http://www.example.org/wine">
  <rdfs:comment>An example OWL ontology</rdfs:comment>
  <owl:versionInfo>
      $Id: Overview.html,v 1.2 2002/11/08 16:42:25 connolly Exp $
  </owl:versionInfo>
  <owl:imports rdf:resource=
  "http://www.w3.org/TR/2002/WD-owl-guide-20021104/food.owl"/>
```

A simple class `Wine` sub-class of `PotableLiquid`. The `rdfs:label` entry provides an optional human readable name for this class.

```
<owl:Class rdf:ID="Wine">
  <rdfs:subClassOf rdf:resource="#PotableLiquid"/>
  <rdfs:label xml:lang="en">wine</rdfs:label>
  <rdfs:label xml:lang="fr">vin</rdfs:label>
  ...
</owl:Class>
```

In addition to classes there are also their members. We normally think of these as *individuals* in our universe of things.

```
    <Region rdf:ID="CentralCoastRegion" />
```

Some words are needed to better explain the difference between class and individual. A ***class*** is simply a name and collection of properties that describe a set of individuals, and ***individuals*** are the members of those sets. Thus classes should correspond to naturally occurring sets of things in a domain of discourse, and individuals should correspond to actual entities that can be grouped into these classes.

Properties let us assert general facts about the members of classes and specific facts about individuals. A property is a binary relation. Two types of

properties are distinguished: *datatype properties*, relations between elements of classes and XML datatypes; *object properties*, relations between elements of two classes. When we define a property there are a number of ways to restrict the elements of the relation. The domain and range can be specified. The property can be defined to be a specialization (subproperty) of an existing property.

```
<owl:ObjectProperty rdf:ID="madeFromGrape">
  <rdfs:domain rdf:resource="#Wine"/>
  <rdfs:range rdf:resource="#WineGrape"/>
</owl:ObjectProperty>
```

The ontology header definition is closed with the following tag.

```
</owl:Ontology>
```

This prelude is followed by the actual definitions that make up the ontology and is ultimately closed by

```
</rdf:RDF>
```

Many other elements can be introduced to enrich the ontology. You can define complex properties (symmetric, transitive, inverseOf, functional, inverseFunctional), restrictions, cardinality, ontology mapping, but the description of them is out of the objective of this thesis. For further information it is possible to visit the W3C web site. It's important to note that the language is steadily developing and changes are frequent.
Here you can find a guide to OWL: `http://www.w3.org/TR/owl-guide/` and here a technical reference: `http://www.w3.org/TR/owl-ref/`

6.2 OntoWeb

OntoWeb[3]: Ontology-based information exchange for Knowledge Management and Electronic Commerce. **OntoWeb** is a *Thematic Network* funded by the European Commission. It started in June 2001 and it will end in May 2004. Its contract number is IST-2000-29243. The Network is coordinated by The Free University of Amsterdam (The Netherlands) and more than one hundred partners from academic and industrial area collaborate to the research. The University of Modena and Reggio E. is a member too.

[3]http://www.ontoweb.org/

As we have seen in the previous section the Web is evolving towards a new generation based on the semantics. To do this, we need ontologies that provide shared and common domain theories. They can be seen as meta data that explicitly represent semantics of data in machine-processable way. Ontology-based reasoning services can operationalize this semantics to provide various services. Ontologies help people and computers access the information they need, and effectively communicate with each other. The goal of the OntoWeb Network is to bring researchers and industry together to promote interdisciplinary work on the Semantic Web.

Another important aspect the project relies on, is the competitiveness of companies active in this area. The business world has a high rate of change so it is of crucial importance how effectively the companies acquire, maintain, exchange and can access to their knowledge, and whether they can deliver the right information to the right individual or business at the right time wherever they are. Due to the impact of the Internet, many organizations are more and more geographically dispersed and organized around virtual teams. Such organizations need knowledge management and organizational memory tools that encourage users to understand each other's changing contextual knowledge and encourage collaboration while capturing, representing and interpreting the knowledge resources of their organizations. At the same time, competitiveness of companies will also depend on the products and services they offer. The growth of a wide range of e-commerce services, both to individuals and between businesses, is contributing to the increase in international trading of products and services. The ability to find, interrogate and exchange knowledge is fundamental for B2B and B2C e-Commerce. OntoWeb will identify and publicize the ontolgy-based technologies required for the promotion of knowledge management and e-Commerce.

The **OntoWeb Network** *will be the European focal point to bring together activities in the area of ontology-based methods and tools for the Semantic Web.* It wants to help companies in the emerging business field by:

- Visioning and outlining the current state-of-the-art of the ontology-related fields in Europe and World Wide and providing a strategic guide to industrial and commercial applications.

- Cooperating with several standardization bodies to promote the development of Ontology-based Content Standardization and Ontology Language Standardization.

- Providing non-discriminatory access to services by individuals and businesses and helping in finding and extracting information from an exponentially growing network.

- Organizing dissemination workshops, special interesting groups, a scientific journal, and training or education courses with special emphasis on Web based applications, e-commerce, knowledge management and information integration. It addresses aspects of know-how transfer between academia and industry about these emerging ontology-based technologies in order to gain competitive advantage in these rapidly developing markets.

The project's members are divided into **Special Interest Groups** (**SIGs**). They study a particular aspect of the project and produce deliverables to explain their outcomes. The OntoWeb SIGs are the following:

- ***Content Standards*** (**SIG1**). It is coordinated by Nicola Guarino (LADSEB-CNR, Italy). This group cooperates with current initiatives related to ontology-based content standardization, develops a framework for characterizing and comparing content standardization efforts, pushes the current standardization initiatives towards well-defined ontology-based harmonization goals, promotes the research on foundational aspects of ontology development and stimulates the transfer of research on ontology development from academia to industry.

- ***Ontology Language Standards*** (**SIG2**). It is coordinated by Ian Horrocks (University of Manchester, UK) and Frank van Harmelen (Vrije Universiteit Amsterdam). The mission of this group is to cooperate with related initiatives (DAML) and standardization efforts and working groups (W3C ontology language standardisation processes). To disseminate the results and transfer the research and needs between academia and industry. To provide a forum for cooperation in the development of language extensions and the initiation of further standardization efforts.

- ***Enterprise-Standard Ontology Environments*** (**SIG3**). The responsible persons for this SIG are Asunción Gómez-Pérez (University of Marid, Spain) and Mike Brown (SemanticEdge, Germany). They try to support a dialog for the development of Quality Requirements for enterprise-standard ontology environments. They promote the effective application of ontology technology in modern enterprises' information technology systems and provide an open forum to promote synergy and cooperation amongst the ontology technology providers and industrial ontology users.

- ***Industrial Applications*** (**SIG4**). It is guided by Hans Akkermans (Vrije Universiteit Amsterdam, the Netherlands). They provide an

open forum to bridge the potential ontology-related researches with the current electronic commerce market, for instance, B2C, B2B.

- ***Language Technology in Ontology Development and Use*** (**SIG5**). Paul Buitelaar, Thierry Declerck and Gunter Neumann (DFKI-Language Technology, Germany) are responsible for this groups. Their main objective is to define an information portal on language technology in ontology development and use (as part of the OntoWeb portal). The portal will serve as a platform for cooperation between R&D groups in language technology and knowledge management, in order to establish a common understanding of market and research needs, of relevant emerging standards, of tools and resources and related benchmarking and evaluation efforts, and to identify priorities in NLP (Natural Language Processing) development for ontology learning and application. Finally, the portal will function also as a contact point for related user and technology communities, among which in particular the "semantic technology" industry.

Further information about the thematic network activities can be found on the web portal where there are the published deliverables:
URL: http://ontoweb.aifb.uni-karlsruhe.de/About/Deliverables

During this thesis we exploit in particular Deliverable D1.3 that presents a complete survey on ontologies tools (see sec. 4.3). An interesting area of the portal is called "*browse ontology*" where you can navigate the community's ontology and search documents or information inserting preferences into the search engine. Moreover a web portal provided by the University of Madrid allows the user to find a lot of information and links of other interesting projects, tools, ontology and applications.
URL http://babage.dia.fi.upm.es/ontoweb/wp1/OntoRoadMap/index.html

Finally we can say that this group is very active and their web portal is an important and updated reference point where a lot of information and links about the Semantic Web and Ontologies can be easily found.

6.3 On-To-Knowledge

On-To-Knowldege[4] is the name of an European project that aims at creating innovative tools for knowledge management. It is funded by the European

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

Commission and is comprised within the EU Information Society Technologies Programme (EU-IST-10132). It started in January 2000 and terminated in June 2002. The project's partners were: Free University of Amsterdam (NL), coordinator, British Telecom (UK), Swiss Life (CH), AIdministrator (NL), CognIT (NO), EnerSearch (SE), AIFB University of Karlsruhe (D), OntoText Lab. (BU).

Information Technology (IT), and especially the Internet/WWW, have boosted the potential for knowledge acquisition and sharing but information resources are heterogeneous, distributed, semi-structured, and enormous in size. So there is the need for tools of selective semantic (meaning-oriented) access.

On-To-Knowledge *wants to improve access to information sources and to lower costs to provide and maintaining large bodies of textual and semistructured information.* In order to reach that objectives they prepared tools for ontology construction and interchange, they built up infrastructure for information representation and they refined the means for information access and extraction. During the project they analyzed three large case studies in order to develop the technology according to the actual needs of large and/or virtual organizations.

The project is now over and in the last deliverable[5] (available on the web site) they describe their achievements. **On-To-Knowledge** provides several breakthroughs on the way to the next generation of the web: advanced tools, an ontology language standard for the web, case studies, and knowledge management methodology.

Now we summarize some of the principal results. Figure 6.1 shows the On-To-Knowledge architecture.

Tools.

A major objective of the project was to create intelligent software to support users in both having access to information and in the maintenance, conversion, and acquisition of information sources. These tools are based on a three-layer architecture around information access, information storage, and information generation.

QuizRDF is an ontology-based tool for knowledge discovery which combines traditional keyword querying of WWW resources with the ability to browse and query against RDF annotations of those resources. RDF(S) and RDF are used to specify and populate an ontology and the resulting RDF anno-

[5] On-To-Knowledge Final Report

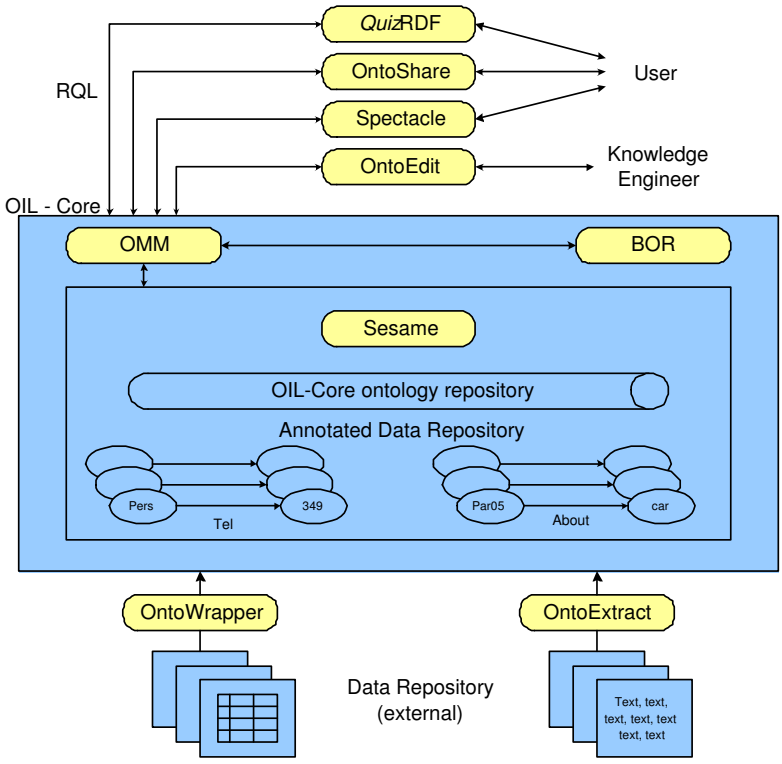

Figure 6.1: On-To-Knowledge architecture

tations are then indexed along with the full text of the annotated resources. The resulting index permits both keyword querying against the full text of the document and against the literal values occurring in the RDF annotations, along with the ability to browse and query the ontology. This ability to combine searching and browsing behaviors supports a typical information-seeking task more fully than "traditional" search engine technology.
At the link below you can find the starting web page from where you can perform searches. The tool requires only a web browser and a network connection to have access to QuizRDF server. After entering a textual query, the users are presented with a set of concepts from which they can select in order to significantly narrow down their search results to a set that is relevant to them. It can either be used in conjunction with an RDF repository such as *Sesame* or it can be used in it is own right to extract RDF from source pages. QuizRDF is a fully developed prototype and has been used in On-To-Knowledge case studies. It has been realized by

BT Exact Technologies[6].
A detailed paper is "J. Davies, U. Krohn, and R. Weeks, *QuizRDF: search technology for thesemantic web.* In WWW2002 workshop on RDF & Semantic Web Applications, 11th International WWW Conference WWW2002, Hawaii, USA, 2002".

URL: http://i97.labs.bt.com/quizrdf-bin/rdfsearch/pmika2.89

OntoShare enables the storage of the best practice information in an ontology and the automatic dissemination of the new best practice information to relevant co-workers. It also allows users to browse or search the ontology in order to find the most relevant information to the problem that they are dealing with at any given time. The ontology helps to orientate new users and acts as a store for key learning and best practices accumulated through experience.
In other words, OntoShare is a knowledge sharing system that allows a community to share information. Users can enter information from the Web or Intranet pages as well as plain text articles. The system then notifies other users that might be interested (based upon user profiles) that some data has been added. OntoShare is a Java application and users only need a web browser to use it. OntoShare has been realized by BT Exact Technologies. The starting page is at the link below.
A detailed paper is "J. Davies, A. Duke, and A. Stonkus, *OntoShare: Using Ontologies for Knowledge Sharing.* In Proceedings of the WWW2002 Semantic Web workshop, 11th International WWW Conference WWW2002, Hawaii, USA, 2002".

URL: http://i97.labs.bt.com/ontoshare/ologin

Spectacle organizes the presentation of information. This presentation is ontology driven. Ontological information, such as classes or specific attributes of information, is used to generate exploration contexts for users. An exploration context makes it easier for users to explore a domain. The context is related to certain tasks, such as finding information or buying products.
At present we have found on the site of AIdministrator[7], the developer of Spectacle, three specialized applications: *Spectacle e-commerce*, *Spectacle analysis*, *Spectacle enterprise*.

[6]http://www.btexact.com/
[7]http://www.aidministrator.nl/

Spectacle e-commerce is a development platform for the creation of web site applications that shows the products of a virtual store helping visitors to find products they really want by sharing knowledge about the products in the store. Spectacle makes it possible for visitors to explore a virtual store in order to learn what is there. Following the link below there is a web site demo.
Spectacle enterprise is dedicated to companies with valuable information repositories that want to keep track of the latest information by automatically classifying new information. This enables workers to find and share information. The process of adding new information to the system is simple with the help of automatic classification software. In this way different users have the information they need presented in the most useful manner, giving them the possibility to save time in finding information.
Spectacle analysis provides the qualitative and quantitative graphical analysis of information, in particular it helps information professionals understand their information collections. Spectacle analysis visually shows the cohesion within large collections of information, typically collections of documents. It integrates structured and unstructured information from document repositories, web sites, databases, and other applications. Figure 6.2 shows the output of the Spectacle Cluster Map Viewer.
A detailed paper is "*Spectacle*, Ch. Fluit, H. ter Horst, J. van der Meer, M.

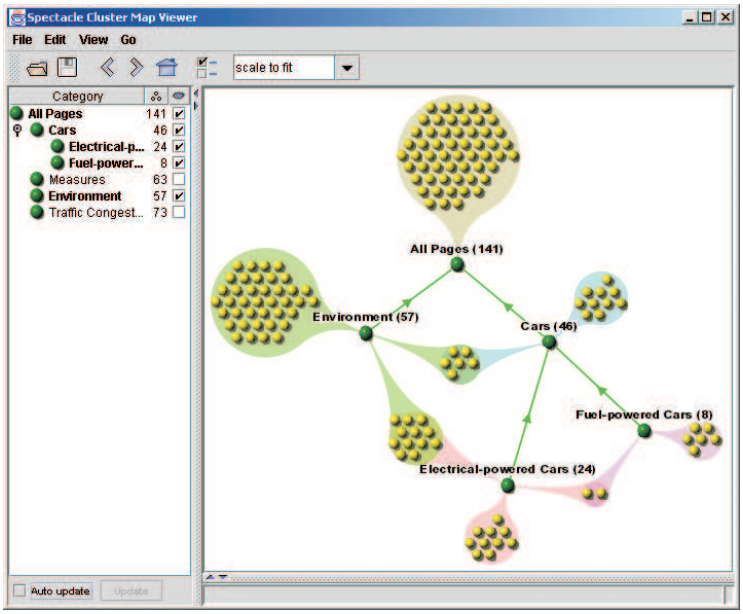

Figure 6.2: Spectacle Cluster Map Viewer.

Sabou,P. Mika, Ch. 9 (pgs. 145-160) of *Towards the Semantic Web:Ontology-driven Knowledge Management*, J. Davies, D. Fensel, F. vanHarmelen (eds.), John Wiley, 2002" or Deliverable D13 of On-To-Knowledge[8].

URL: http://www.aidministrator.nl/spectacle/channel
/summer2002?section=products/spectacle

OntoEdit makes it possible to inspect, browse, codify and modify ontologies, and thus serves to support the ontology development and maintenance task. Modelling ontologies using OntoEdit involves modelling at a conceptual level, that is: (i) as independently of a concrete representation language as possible, and (ii) using GUI's representing views on conceptual structures (concepts, concept hierarchy, relations, axioms) rather than codifying conceptual structures in ASCII.
This tool has been developed in java by Ontoprise (The Ontoprise GmbH was founded 1999 as a spin off of the University of Karlsruhe) and it is available in different versions; one of them is downloadable free from the web site (http://www.ontoprise.de/customercenter/software_downloads/free_en), other versions provide more services and plugins to enrich its features.
A detailed paper is "Y. Sure et al.: OntoEdit: Collaborative Ontology Engineering for the Semantic Web. In Proceedings of the first International Semantic Web Conference 2002 – ISWC 2002" or Deliverable D3 of On-To-Knowledge[9].
Further information has been presented also in sec. 4.3.1.

URL: http://www.ontoprise.de/ontoedit.htm

Ontology Middleware Module (**OMM**) supports well-defined application programming interfaces (OMAPI) used for access to knowledge and deals with such matters as:

- Ontology versioning, including branching;
- Security – user profiles and groups are used to control the rights for access, modifications and publishing;
- Meta-information and ontology lookup – support for meta-properties (such as Status, Last-Updated-By, Responsible, Comments, etc.) for whole ontologies, as well as for separate concepts and properties. Ontology and concept lookup according to meta-information is possible;

[8]http://www.ontoknowledge.org/downl/del13.pdf
[9]http://www.ontoknowledge.org/countd/countdown.cgi?del3.pdf

- Access via a number of protocols: HTTP, RMI, EJB, CORBA, and SOAP.

The Ontology Middleware Module is an enterprise back-end for formal knowledge management. It permits to knowledge representation and management in RDF(S) and languages structurally compatible with it, such as, DAML+OIL and OWL. OMM is an extension of Sesame.
Further information can be found in Deliverable D38[10].
The Ontology Middleware is open source and is available for download and contributions at http://sourceforge.net/projects/sesame/
It has been developed by the OntoText Lab[11]. On the web site a working demo version is also present.

URL: http://www.ontotext.com/omm

Sesame is a system that permits persistent storage of RDF data and schema information and subsequent online querying of that information. One of the most important modules of Sesame is the query engine. This query engine supports an OQL-style query language called RQL. RQL supports querying of both RDF data (e.g. instances) and schema information (e.g. class hierarchies, domains and ranges of properties). RQL also supports path-expressions through RDF graphs, and can combine data and schema information in one query.
Sesame has been developed in java by AIdministrator and it needs to be installed locally; it is open source and can be downloaded at
http://sourceforge.net/projects/sesame/
Further information can be found here: "Jeen Broekstra, Arjohn Kampman, Frank van Harmelen(2002). *Sesame: A generic architecture for storing and queryingrdf and rdf schema.* In Proceedings of the First International SemanticWeb Conference, Horrocks, I. & Hendler, J. A. (Eds.), volume 2342 of Lecture Notes in Computer Science. Springer"
http://www.cs.vu.nl/ frankh/postscript/ISWC02.pdf
On the Sesame web site several documents and also a demo version of the system are available.

URL: http://sesame.aidministrator.nl

BOR provides additional reasoning services so as to extend the functionality provided by SESAME. Most of the classical reasoning tasks for description

[10]http://www.ontoknowledge.org/downl/del38.pdf
[11]http://www.ontotext.com/

logics (DL) are available, including realization and retrieval.
In other words BOR is a description logics (DL) reasoner. It has been developed by OntoText in java and it has been integrated in the Sesame system. It can be downloaded from the web site with examples and tutorials. Like Sesame, it needs to be installed locally.
On-To-Knowledge has studied BOR in Deliverable D40
(http://www.ontoknowledge.org/downl/del40.pdf).

URL: http://www.ontotext.com/bor

CORPORUM toolset (**OntoWrapper** and **OntoExtract**) performs information extraction and ontology generation. It is situated in the extraction layer. CORPORUM has two related tasks: interpretation of natural language texts and extraction of specific information from free text. Whereas CORPORUM tools can perform the former process autonomously, the latter task requires a user who defines business rules to extract information from tables, (phone) directories, homepages, etc.
These tools allow the user to specify the pages he wants to analyze specifying many preferences to refine the final result (an ontology). Tools have been developed as Microsoft COM objects by CognIt[12] and need to be installed locally. Following the link below you can find further documents and tutorials.
A specific paper is "R.Engels and B.A. Bremdal. *CORPORUM: A Workbench for the Semantic Web*. Semantic Web Mining workshop. PKDD/ECML - 01. Freiburg, Germany,2001".

URL: http://ontoserver.cognit.no/

Language.

Regarding the language, **OIL language** has been studied (cf. D. Fensel et al.: OIL: Ontology Infrastructure to Enable the Semantic Web, IEEE Intelligent System, 2001). It was designed to combine frame-like modelling primitives with the increased expressive power, formal rigor and automated reasoning services of an expressive description logic. OIL is "web enabled" by having both XML and RDFS based serializations. As part of the Semantic Web activity of the W3C, a very simple web-based ontology language had already been defined, namely RDF Schema. This language only provides facilities to define class and property names, inclusion axioms for both classes

[12]http://www.cognit.no/

and properties (subclasses and sub properties), and to define domain and range constraints on properties. *OIL has been designed to be a superset of the constructions in RDF Schema and the syntax of OIL has been designed such that any valid OIL document is also a valid RDF(S) document when all the elements from the OIL-namespace are ignored.*

For many of the applications, it is unlikely that a single language will be ideally suited for all uses and all users. In order to allow users to choose the expressive power which is appropriate to their application, and to make future extensions possible, a layered family of OIL languages has been constructed. The sub-language *OIL Core* has been defined to be exactly the part of OIL that coincides with RDF(S). This corresponds to full RDF(S), without some of RDFs more dubious constructions: containers and reification.
During the project it was observed that *some problems arise using OIL*. In fact in the case studies they worked on, the domain experts often did not employ the features of OIL to construct ontologies, they simply referred to RDF(S) or to Core-OIL. Very lightweight ontologies were constructed, sometimes consisting of not much more than taxonomies of terms, often with no properties relating them. Similarly, the automatic concept-extraction technology that was employed also yielded very light-weight collections of terms, with almost no hierarchical structure and with only very general associations between the concepts (of the kind: "Concept1 isStronglyRelatedTo Concept2"). They try to explain these problems and they state that:

- Some case studies already started even before the first version of OIL was available;
- OIL lacks any tutorial support, any customized tool support, and any real practical experience. From a user point of view this makes it nearly unusable for the moment;
- OIL is ongoing standardization efforts including high change rates. Neither the user nor the tool developer can trust the current language version as being final and not just an intermediate step. Using OIL is of high risk, both for users in case studies as well as for tool developers.

Methodology.

Finally, a methodology for employing ontology-based tools for knowledge management applications was developed. They prepared an initial baseline methodology and applied it to the case studies they worked on. This

methodology provides guidelines for introducing knowledge management concepts and tools into enterprises, helping knowledge providers and seekers to present knowledge efficiently and effectively.

At the end of the project they published a book to present and promote their outcomes ("Towards the Semantic Web. Ontology-Driven Knowledge Management". J. Davies, D. Fensel, F. van Harmelen, Wiley ed.). They contributed to the creation of the first International Semantic Web Conference (ISWC) that took place in June 2002 in Sardinia (Italy) and they will organize the second one in October 2003 in Florida (USA).[13]

The researchers that worked at the On-To-Knowledge project are strongly involved in creating the international standard called OWL described before. In co-operation with the large US project DAML[14] they developed a joined consensual sub-dialect of OIL called DAML+OIL.

6.4 MAFRA

MAFRA – A Mapping FRAmework for Distributed Ontologies [33] developed at the University of Karlsruhe. *It is an interactive, incremental and dynamic framework for mapping distributed ontologies.*

As we have seen also before, the de-centralized nature of the Web makes indeed inevitable that communities will use their own ontologies to describe their data. In this vision, ontologies are themselves distributed and the key point is the mediation between distributed data using mappings between ontologies. Thus, complex mappings and reasoning about those mappings are necessary for comparing and combining ontologies, and for integrating data described using different ontologies.

Following the MAFRA approach, *an ontology mapping process is the set of activities required to transform instances of a source ontology into instances of a target ontology.* The framework, they are building, consists of five horizontal modules describing the fundamental phases of mapping process. Four vertical components run along the entire mapping process, interacting with horizontal modules (see Figure 6.3).

Within the horizontal dimension, there are the following five modules:

Lift and Normalization. This module focuses on raising all data to be mapped onto the same representation level, coping with syntactical, structural and language heterogeneity. Both ontologies must be normalized to a

[13] http://iswc.semanticweb.org/
[14] http://www.daml.org/

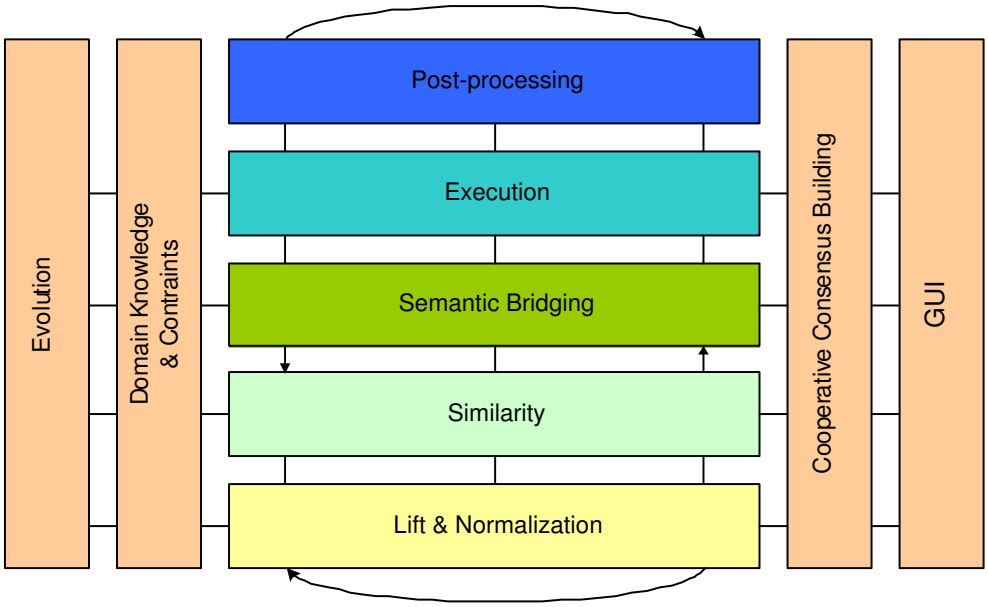

Figure 6.3: MAFRA conceptual architecture

uniform representation, that is RDF(S), thus eliminating syntax differences and making semantics differences between the source and the target ontology more apparent. The source and target ontologies are already represented in RDF-Schema with their instances in RDF. An essential step of this first phase is also normalization. Three distinct ordered tasks are performed in our approach: (i) tokenization of the entities, (ii) elimination of resulting stop words and (iii) expansion of acronyms. The result is a list of normalized lexicon.

Similarity. This module establishes similarities between entities from the source and target ontology, thus, it supports mapping discovery. Several different similarity measures have been proposed in literature. They adopted a multi-strategy process, that calculates similarities between ontology entities using different algorithms.

Semantic Bridging. Based on the similarities computed in the previous phase, the semantic bridging phase is responsible for establishing correspondence between entities from the source and target ontology. It intends to specify bridges between entities in a way that each instance represented according to the source ontology is translated into the most similar instance described according to the target ontology.

Execution. This module actually transforms instances from the source ontology into target ontology by evaluating the semantic bridges defined earlier. In general two distinct modes of operation are possible, namely offline (static, one-time transformation) and online (dynamic, continuous mapping between source and the target) execution.

Post-processing. The post-processing component takes the results of the execution module to check and improve the quality of the transformation results. The most challenging task of post-processing is establishing object identity-recognizing that two instances represent the same real-world object.

The vertical dimension of MAFRA contains modules that interact with horizontal modules during the overall mapping process. Four modules have been identified.

Evolution. This aspect focuses on keeping semantic bridges obtained by the "Semantic Bridge" module, which must be kept in synchrony with the changes in the source and target ontologies.

Cooperative Consensus Building. The cooperative consensus building aspect is responsible for establishing a consensus on semantic bridges between two communities participating in the mapping process. This is a requirement since one has to choose frequently from multiple, alternatively possible mappings .The amount of human involvement required to achieve consensus may be reduced by automating the mapping process as much as possible.

Domain Constraints and Background Knowledge. The quality of similarity computation and semantic bridging may be dramatically improved by introducing background knowledge and domain constraints, e.g. by using glossaries to help identify synonyms or by using lexical ontologies, such as WordNet or domain-specific thesauri, to identify similar concepts.

Graphical User Interface. Mapping is a difficult and time consuming process, which is not less difficult than building an ontology itself, i.e. deep understanding of both conceptualizations required on human side, thus extensive graphical support must be given and it is a separate issue how this can be achieved in an optimal way.

MAFRA is currently under development within the **KAON Ontology and Semantic Web Framework**. **KAON**[15] is the Karlsruhe Ontology and Semantic Web framework, it has been developed at the University of Karlsruhe (Germany) and is used as a basis for several ontology-enabled research and industry projects. Its primary goal is to establish a platform needed to apply Semantic Web technologies to e-commerce scenarios, knowledge management, automatic generation of Web portals, E-Learning etc. Till now they achieved the implementation of four modules of MAFRA: the automatic similarity discovery module, the semantic bridging representation, the graphical user interface and the execution engine.

Navigating the KAON web portal you can find all the updated information about the evolution of modules (http://kaon.semanticweb.org/modules) and download them (http://kaon.semanticweb.org/download). Tools have been developed in Java. There is also an interesting section of the site where it is possible to find demos of KAON (http://kaon.semanticweb.org/demos). Moreover there are several useful ontologies to download for testing or using with tools (http://kaon.semanticweb.org/ontologies). Further information such as tutorials, papers, and links to other projects that use KAON are available in the *documentation, papers* and *projects* sections of the site.

6.5 SHOE

The **Simple HTML Ontology Extension** (**SHOE**[16]) *is an HTML-based knowledge representation language* developed at the Computer Science Department of the Maryland University. SHOE is a superset of HTML which adds the tags necessary to embed arbitrary semantic data into web pages [34].

The underlying philosophy of SHOE is that intelligent internet agents will be able to better perform their tasks if the most useful information is provided in a structured manner. To this end, *SHOE extends HTML* with a set of knowledge oriented tags that, unlike HTML tags, provide structure for knowledge acquisition as opposed to information presentation. *SHOE associates meaning with this content by making each web page commit to one or more ontologies.* These ontologies permit the discovery of implicit knowledge through the use of taxonomies and inference rules, allowing information providers to encode only the necessary information on their web pages, and to use the level of detail that is appropriate to the context. Interoperability

[15] http://kaon.semanticweb.org
[16] http://www.cs.umd.edu/projects/plus/SHOE/index.html

is promoted through the sharing and reuse of ontologies. To achieve compatibility with existing web standards, SHOEs syntax is defined as an application of SGML, a language that defines tag-based languages and was the influence for HTMLs syntax. A slight variant of the syntax exists for compatibility with XML, and can be used by web sites that are migrating to XML.

SHOE tags are divided into two categories. First, there are *tags for constructing ontologies*. SHOE ontologies are sets of rules which define what kinds of assertions SHOE documents can make and what these assertions mean. For example, a SHOE ontology might say that a SHOE document can declare that a data entity is a "dog", and that this "dog" is permitted to have a "name". Secondly, there are *tags for annotating web documents to subscribe to one or more ontologies, declare data entities, and make assertions about those entities under the rules proscribed by the ontologies*. For example, a SHOE document subscribing to the SHOE ontology above, might then declare that it is all about a dog named "Fido".

SHOE can be used to embed data from a variety of sources and for a variety of purposes. It is not intended for any particular function. However, SHOE is primarily meant to make it possible for web robots and intelligent agents to finally simplify the actions we perform every day.

SHOE is not just a meta-content language. SHOE provides a relatively rich level of semantics and abilities, which enable web designers to embed documents not only with information about the overall "content" of those documents but any arbitrary information at all. SHOE also allows agents to make automatic inferences about the data they learn, provides a hierarchical categorization scheme, and a sophisticated ontology mechanism designed specifically for the web needs. SHOE tags can be used for a wide range of agent-based functions.

SHOE does not have any pre-defined ontologies, categories, relationships, or inferences. SHOE is a language in which categories, relationships, attributes, inferences, etc. can be defined by ontologies, but SHOE itself does not define them. This is the job of ontology designers for specific tasks or domains. However, the SHOE project offers some initial ontologies to start. SHOE Ontologies declare:

- Classifications (categories) for data entities. Classifications may inherit from other classifications ("Dogs are Animals").
- Valid relationships between data entities and other data entities or simple data (strings, numbers, dates, boolean). Arguments for relation-

ships are typed, either by the simple data that can fill the argument, or with the classification a data entity must fall under in order to fill an argument ("Dogs can chase cats").

- Inferences in the form of horn clauses with no negation ("If a person works for an organization, that person automatically works for any organization the organization is a sub-part of").
- Inheritance from other ontologies: ontologies may be derived from or extend zero or more outside ontologies ("The SPCA ontology extends the common Library of Congress ontology").
- Versioning. Ontologies may extend previous ontology versions.

HTML pages with embedded SHOE data may:

- Declare arbitrary data entities. Usually, one of these entities is the web page itself.
- Declare the ontologies which they will use when making declarations about entities ("I'm using the 'Pets' ontology promulgated by the SPCA").
- Categorize entities ("This entity is a dog").
- Declare relationships between entities or between entities and data ("This entity likes to chase that entity", "This entity's name is 'Fido"').

SHOE permits n-ary relations, simple inheritance in the form of classification, multi-valued relations, and a conjunctive knowledge base. It does not currently permit negation, disjunction, or arbitrary functions and predicates. In order to add SHOE to web pages there are two methods: I) you can use the "Knowledge Annotator" that is a tools developed to help user that are not familiar with the HTML and the SHOE language; II) you can add by yourself the right tags into your web (html) pages according to SHOE specifications (http://www.cs.umd.edu/projects/plus/SHOE/spec.html). Here we show a simple example of a SHOE ontology.

```
<HTML>
...
<BODY>
<ONTOLOGY ID="cs-dept-ontology" VERSION="1.1"
                BACKWARD-COMPATIBLE-WITH="1.0">
<USE-ONTOLOGY ID="univ-ontology" VERSION="1.0" PREFIX="u"
                URL="http://ontlib.org/univ_v1.0.html">
```

```
...
<DEF-CATEGORY NAME="ComputerScience" ISA="u.ResearchArea">
...
<DEF-RELATION NAME="writtenIn">
    <DEF-ARG POS=1 TYPE="Program">
    <DEF-ARG POS=2 TYPE="ComputerLanguage">
</DEF-RELATION> ... <DEF-RENAMEFROM="u.Department"TO="Department">
<DEF-RENAME FROM="u.Chair" TO="DepartmentHead">
...
</ONTOLOGY>
</BODY>
</HTML>
```

The researchers of SHOE have recently moved to the **MIND SWAP**[17] (Maryland Information and Network Dynamics Lab - Semantic Web Agents Project) of the University of Maryland. They use OWL and DAML+OIL languages that are partially based on SHOE.

On both web sites (SHOE and MIN SWAP) different tools are available to download: web pages annotator (SMORE, Knowledge Annotator, RIC), converter (ConvertToRDF, OWL Converter, Excel2RDF). The tools have been developed using Java. Moreover they have annotated web pages and they propose a search engine that performs semantic searches exploiting SHOE features.

6.6 Comments

We begin observing that many projects are on going with respect to the new issue of the Semantic Web but we think that there is not a clear and common objective; in some way, this fact creates some confusion in the research environment. Probably the causes of such a situation are that, on one hand, the idea of Semantic Web has created a lot of expectations but on the other hand this idea has not specified what it is necessary to become concrete. Thus the research has followed different trails instead of concentrating on few clear requirements or components to build the Semantic Web.

The projects and the studies that OntoWeb and On-To-Knowledge are conducting, are heterogeneous. Specially the tools that On-To-Knowledge has developed, cooperate to realize their architecture, but they do not form

[17] http://www.mindswap.org/

a complete environment for the management of all the aspects related to the dynamics of ontologies. Moreover the tools have been developed by many research groups and therefore they are quite different.

From our point of view this fragmentation is a problem, because our first objective is to build a complete infrastructure that can lead the users during all the life cycle of ontologies, from their creation, through their exploitation, their evolution, their interrogation. Concerning this we can say that the study of related projects has been useful to acquire important knowledge about the management of ontologies and to elaborate our proposal.

With respect to the results of MAFRA we can observe that the approach they have chosen is very interesting. They try to build a framework that helps the users in all the phases of ontologies' management. We remember that they use a mapping approach to solve the problem of distributed ontologies, while our objective is to construct a common and shared ontology and use this integrated view to look at the represented domain.

Finally we have languages. As we have seen before, On-To-Knowledge researchers have encountered problems using the new language they studied, OIL, and they have tried to explain some causes. We are convinced that also SHOE suffers the same problems. In fact, the researchers have to be sure that the language is stable or better, standard in order to develop tools and models that can rely on it. Moreover both OIL and OWL (its successor) have more than one level of use; this flexibility is necessary to help domain experts providing them with the most suitable level of details, during the modelling phase. Probably researchers will change their language only when a new language will reach a sure status of standardization and the improvements will be tested clearly.

In consequence of these drawbacks related to new languages, we decided to keep ODL_{I^3} and XML as base language to represent and to export our data.

Conclusions

In this work we have studied the following issues:

- the Semantic Web and the new Web scenarios;
- ontologies and their management;
- the MOMIS approach for supporting dynamics of ontologies.

One of the main objectives of this work has been the understanding of the scenarios and the problems introduced by ***the Semantic Web***. Many research groups are trying to find an effective solution but it is an hard task. The infrastructure of the web is not ready yet to realize the useful expectations of this new web.
In particular, a lot of work has been done in finding a solution for ontologies management. We analyzed the many tools which try to face some aspects of this issue, but there is not any tool that solves the problem in its completeness and complexity. Moreover, the rapid evolving of domains and web environments is a further difficult problem to be faced.

We can say that the Semantic Web vision is very attracting because its principal aim is to simplify the way in which people exploit the Web, but, studying the problem, we understood that there is not a clear research line to follow yet. The scientific community, in our opinion has to work hard in order to define solutions.

A good point in this direction is that W3C is very active in this new research area; an evidence of this interest is the creation of a special group with the task to explore the problems related to the Semantic Web. They are charged to develop facilities to put machine-understandable data on the Web.

Strictly related to these issues, we tried to understand the role that ***ontologies*** have in building of the Semantic Web.

We have analyzed two fundamental aspects of ontologies: ***languages*** and ***dynamics***.

With respect to ***languages*** we have analyzed OWL and SHOE. OWL (Ontology Web Language) has been developed by W3C and it is based on RDF, while SHOE (Simple HTML Ontology Extension) has been developed at the Computer Science Department of the Maryland University. SHOE is a superset of HTML which adds the tags necessary to embed arbitrary semantic data into web pages. The semantic enrichment process and the annotation of web pages with respect to a reference ontology is the basic need to permits users to perform more precise web searches.
OWL and SHOE languages are not yet "standard" and, moreover, they are under development, thus researchers have not yet based their tools on these languages, but they have preferred the safer RDF, XML, etc.

Dynamics is the other aspect of ontologies that we tried to face. The causes of ontologies change are numerous: evolution in the represented domain, a different point of view, errors during the modelling phase, new objectives, etc. In literature we have found two main approaches: the *evolution approach* and the *versioning approach*. The first one is very interesting because it tries to solve the problem in its completeness and complexity, aiming to keep the ontology consistent. The second approach, instead, tries to manage different versions of an ontology providing a methodology to refer to the right version during the ontology use.

In order to acquire important knowledge for the improvement of the MOMIS system we have analyzed the principal ***tools for developing and merging or integrating ontologies***. Many tools have been developed by projects such as OntoWeb or On-To-Knowledge, by Universities (Stanford, Madrid, Karlsruhe), or by companies (OntoPrise, AIdministrator). Concerning tools, we can argue that some of them are quite effective, but often they are not inserted in a complete framework that helps the ontology designer in all the situations he has to face to keep consistent and updated the reference ontology.

With respect to the **MOMIS** system we have elaborated *a solution to manage the insertion of new sources in the system thus facing the problem of dynamics of the already built ontoloy*. We have presented some modification to the MOMIS integration process that allow to semantically enrich the global schema and to integrate new sources.

First of all, our approach requires the annotation of the Global Virtual View elements (Global Classes and Global Attributes), created with the MOMIS methodology. The second phase is the integration of the new sources with the previous GVV.

Our proposal aims to preserve as much as possible the results of the previous integration phase because we assume too costly restarting the process from scratch. The annotation phase plays a key role to add the proper and necessary semantics to the schema and moreover in this way we transform it in the best candidate for the integration of new sources. In other words, the new integration process is based on the exploitation of lexicon-derived relationships extracted after the annotation phase among previous global elements and new local elements of the added sources.

At present, the process that we have developed solves only the case of the insertion of new sources. This first step is important because we have understood the key role of the semantic enrichment in order to achieve good results. In the future the system can be extended to manage the modification or the deletion of an integrated source.

The new integration process has been partially implemented in the MOMIS system.

Finally we express some consideration on the importance of our work for the **SEWASIE** project, in fact, MOMIS is a fundamental component of the SEWASIE system. The capability to manage dynamically the insertion of new sources is of great importance for the SEWASIE framework.

The efforts to extend the functionalities of the MOMIS and SEWASIE systems is demonstrated also by the works done in different directions. In the following we report two recent thesis.

Veronica Guidetti in her degree thesis [30] studied the possibility to extend the WordNet lexical system. Her work wants to allow the user of SI-Designer to annotate with the right meanings new terms extracted by sources. This new possibility helps the annotation and semantic enrichment process, increasing the probability to obtain a better integration result in the MOMIS context. Moreover, this work is important because it opens the possibility to extend the system to other languages different from English. Guidetti's achievement will be integrated in the MOMIS system in the near future.

Another way to improve the completeness and the features of MOMIS is the ability to find new sources to be integrated on the web. Doing this, a problem is the format of web pages; HTML, in fact, is human-readable, and so we need wrappers to transform HTML in XML that is a machine-readable format and thus to exploit it within the MOMIS system. Lorenzo Lugli in his thesis [35] reviews some of the most important wrappers available today.

Appendix A

The running example

In this appendix we present a complete running example of the MOMIS framework.

We start introducing a schematic model of two sources (Figure A.1). The sources derive from an university scenario:

- The first source, `University` (UNI), is a *relational DataBase* that contains data about personnel and student of a specific university; it has the following tables: `Research_Staff`, `School_Member`, `Department`, `Section`, `Room` and `Article`. For a given professor (in `Research_Staff`) his department (`dept_code`), his section (`section_code`) and his e-mail address (`e-mail`) are stored. In the relation `School_Member` the information `name`, `year` and `faculty` about students enrolled at the university are stored. `Department` has a code (`dept_code`), a name (`dept_name`) and a `budget`. We know that a `Section` has a code (`section_code`), a name (`section_name`), a `length` and it is related to a room (`room_code`). With reference to `Room` we know its code (`room_code`), the number of seats (`seats_number`) and we have some `notes`. Finally we have articles, written by an `author`, with a `title`, a code (`article_code`), the year of publication and the `subject`.

- The source `Computer_Science` (CS) represents information about people related to the Computer Science department. This source is an *object DataBase*. There are nine classes: `CS_Person`, `Professor`, `Student`, `Office`, `Location`, `ClassRoom`, `Essay`, `Publication` and `Course`. Information is quite similar to the first source: it stores data on *professors* (`Professor`) and *students* (`Student`), also giving the possibility to retrieve the `Office` of a given professor. The class `Location` maintains the position inside the computer science department. With respect to

Sorgente UNIVERSITY (*UNI*)

```
Research_Staff(relation,name,dept_code,section_code,e_mail)
School_Member(name,faculty,year)
Department(dept_name,dept_code,budget)
Section(section_name,section_code,length,room_code)
Room(room_code,seats_number,notes)
Article(year,subject,author,title,journal,article_code)
```

Sorgente COMPUTER_SCIENCE (*CS*)

```
CS_Person(first_name,last_name)
Professor:CS_Person(belongs_to,rank,title)
Student:CS_Person(year,takes,rank)
Office(description,address,code)
Location(floor,number,building_name)
Course(course_name,taught_by,taught_in)
Essay:Publication(subject,journal)
Publication(year,title,authors)
ClassRoom(seats,code,address,notes)
```

Figure A.1: Reference example

students (`Student`), we may know the courses they take (`takes`), their `year` and their `rank`. Moreover we have information about the publications (`Publication`) or essays (`Essay`) written by the computer science's members and of the classrooms (`ClassRoom`) where courses (`Course`) are taught.

Now we present the description of the two sources in ODL_{I^3} language.

Source `University` UNI:

```
interface Research_Staff
( source relational University
  extent Research_Staff
  key (name)
  foreign_key (dept_code) references Department (dept_code)
  foreign_key (section_code) references Section (section_code) )
{ attribute string relation;
  attribute string name;
  attribute long section_code;
  attribute long dept_code;
  attribute string e_mail; };

interface Room
( source relational University
  extent Room
  key (room_code) )
{ attribute long seats_number;
  attribute string notes;
  attribute long room_code; };

interface Article
( source relational University
  extent Articles
  key (article_code)
  foreign_key (author) references Research_Staff (name) )
{ attribute long year;
  attribute string subject;
  attribute string author;
  attribute string title;
  attribute string journal;
  attribute long article_code; };

interface School_Member
( source relational University
  extent School_Member
  key (name) )
{ attribute long year;
  attribute string name;
  attribute string faculty; };

interface Section
( source relational University
  extent Section
  key (section_code)
  foreign_key (room_code) references Room (room_code) )
{ attribute long section_code;
  attribute long length;
  attribute string section_name;
  attribute long room_code; };

interface Department ( source relational University
  extent Department
  key (dept_code) )
{ attribute long dept_code;
  attribute long budget;
  attribute string dept_name; };
```

Source `Computer_Science` CS:

```
interface Professor: CS_Person
( source object Computer_Science
  extent Professors )
{ attribute string rank;
  attribute string title;
  attribute Office belongs_to; };

interface Essay: Publication
( source object Computer_Science
  extent Essays )
{ attribute string subject;
  attribute string journal; };

interface Publication
( source object Computer_Science
  extent Publications )
{ attribute long year;
  attribute set<Professor> authors;
  attribute string title; };

interface Office
( source object Computer_Science
  extent Offices
  key (description) )
{ attribute string code;
  attribute string description;
  attribute Location address; };

interface Student: CS_Person
( source object Computer_Science
  extent Students )
{ attribute long year;
  attribute string rank;
  attribute set<Course> takes; };

interface Course
( source object Computer_Science
  extent Courses )
{ attribute string course_name;
  attribute Professor taught_by;
  attribute ClassRoom taught_in; };

interface Location
( source object Computer_Science
  extent Locations )
{ attribute long floor;
  attribute long number;
  attribute string building_name; };

interface CS_Person
( source object Computer_Science
  extent CS_Persons
  key (first_name, last_name) )
{ attribute string first_name;
  attribute string last_name; };
```

```
interface ClassRoom
 ( source object Computer_Science
   extent ClassRooms )
 { attribute long seats;
   attribute long code;
   attribute Location address;
   attribute string notes; };
```

The *Common Thesaurus* at the end of the sources analysis contains the following relations. The last two column represent the module that has produced the relation and if the relation is validated.

Common Thesaurus relationships

Term	Rel.	Term	Module	Validated
Computer_Science.Location	RT	Computer_Science.ClassRoom	SIMA	YES
University.Department	RT	University.Research_Staff	SIMA	YES
University.Section	RT	University.Research_Staff	SIMA	YES
University.Research_Staff.name	SYN	University.Article.author	SIMA	YES
University.Research_Staff	RT	University.Article	SIMA	YES
University.Room	RT	University.Section	SIMA	YES
Computer_Science.Professor	NT	Computer_Science.CS_Person	SIMA	YES
Computer_Science.Essay	NT	Computer_Science.Publication	SIMA	YES
Computer_Science.Student	NT	Computer_Science.CS_Person	SIMA	YES
Computer_Science.Office	RT	Computer_Science.Professor	SIMA	YES
Computer_Science.Professor	RT	Computer_Science.Publication	SIMA	YES
Computer_Science.Location	RT	Computer_Science.Office	SIMA	YES
Computer_Science.Course	RT	Computer_Science.Student	SIMA	YES
Computer_Science.Professor	RT	Computer_Science.Course	SIMA	YES
Computer_Science.ClassRoom	RT	Computer_Science.Course	SIMA	YES
University.Section	RT	Computer_Science.Professor	SIMB	YES
University.Article	RT	Computer_Science.Professor	SIMB	YES
University.Department	RT	Computer_Science.Professor	SIMB	YES
Computer_Science.Office	RT	University.Research_Staff	SIMB	YES
Computer_Science.Course	RT	University.Research_Staff	SIMB	YES
Computer_Science.Publication	RT	University.Research_Staff	SIMB	YES
University.Room	RT	Computer_Science.Course	SIMB	YES
Computer_Science.Course	RT	University.School_Member	SIMB	YES
Computer_Science.Student	RT	University.Section	SIMB	YES
Computer_Science.ClassRoom	RT	University.Section	SIMB	YES
University.Article	NT	University.Research_Staff.relation	SLIM	YES
University.Article.year	SYN	University.School_Member.year	SLIM	YES
University.Article.year	SYN	Computer_Science.Publication.year	SLIM	YES
University.Article.year	SYN	Computer_Science.Student.year	SLIM	YES
University.School_Member.year	SYN	Computer_Science.Publication.year	SLIM	YES
University.School_Member.year	SYN	Computer_Science.Student.year	SLIM	YES
Computer_Science.Publication.year	SYN	Computer_Science.Student.year	SLIM	YES
University.Article.author	SYN	Computer_Science.Publication.authors	SLIM	NO
Computer_Science.CS_Person	BT	University.Article.author	SLIM	YES
Computer_Science.CS_Person	BT	Computer_Science.Publication.authors	SLIM	YES
University.Research_Staff.name	SYN	University.School_Member.name	SLIM	YES
University.Research_Staff.name	NT	University.Research_Staff.relation	SLIM	YES
University.School_Member.name	NT	University.Research_Staff.relation	SLIM	YES
Computer_Science.Professor.title	NT	University.Research_Staff.name	SLIM	YES
Computer_Science.Professor.title	NT	University.School_Member.name	SLIM	YES
Computer_Science.Professor.title	NT	University.Research_Staff.relation	SLIM	YES
University.Research_Staff	SYN	Computer_Science.Professor	SLIM	YES
University.Research_Staff	NT	Computer_Science.CS_Person	SLIM	YES
Computer_Science.Location.floor	RT	Computer_Science.Location.building_name	SLIM	NO
Computer_Science.Essay	NT	University.Research_Staff.relation	SLIM	YES
Computer_Science.Office.code	NT	University.Research_Staff.relation	SLIM	YES
University.Section	SYN	University.Section.section_name	SLIM	YES
University.Section	SYN	Computer_Science.Course	SLIM	YES
University.Section	SYN	Computer_Science.Course.course_name	SLIM	YES
Computer_Science.Course	SYN	University.Section.section_name	SLIM	YES

Term	Rel.	Term	Module	Validated
University.Section.section_name	SYN	Computer_Science.Course.course_name	SLIM	YES
Computer_Science.Course	SYN	Computer_Science.Course.course_name	SLIM	YES
University.Room	RT	Computer_Science.Location.building_name	SLIM	YES
University.Article.title	SYN	Computer_Science.Publication.title	SLIM	YES
University.Article.title	NT	University.Research_Staff.relation	SLIM	YES
Computer_Science.Publication.title	NT	University.Research_Staff.relation	SLIM	YES
Computer_Science.Professor.rank	SYN	Computer_Science.Student.rank	SLIM	YES
University.School_Member	SYN	Computer_Science.Student	SLIM	YES
University.School_Member	NT	Computer_Science.CS_Person	SLIM	YES
Computer_Science.Office.address	SYN	Computer_Science.ClassRoom.address	SLIM	YES
Computer_Science.Office.address	NT	University.Room.room_code	SLIM	NO
Computer_Science.ClassRoom.address	NT	University.Room.room_code	SLIM	NO
Computer_Science.ClassRoom	NT	University.Room	SLIM	YES
University.Room.notes	SYN	Computer_Science.ClassRoom.notes	SLIM	YES
University.Room.notes	NT	University.Research_Staff.relation	SLIM	YES
Computer_Science.ClassRoom.notes	NT	University.Research_Staff.relation	SLIM	YES
Computer_Science.CS_Person.first_name	NT	University.Research_Staff.name	SLIM	YES
Computer_Science.CS_Person.last_name	NT	University.Research_Staff.name	SLIM	YES
Computer_Science.CS_Person.first_name	NT	University.School_Member.name	SLIM	YES
Computer_Science.CS_Person.last_name	NT	University.School_Member.name	SLIM	YES
University.Article.title	NT	University.Research_Staff.name	SLIM	YES
University.Article.title	NT	University.School_Member.name	SLIM	YES
Computer_Science.Publication.title	NT	University.Research_Staff.name	SLIM	YES
Computer_Science.Publication.title	NT	University.School_Member.name	SLIM	YES
University.Room.seats_number	SYN	Computer_Science.ClassRoom.seats	SLIM	YES
University.Room.seats_number	NT	University.Research_Staff.relation	SLIM	NO
Computer_Science.ClassRoom.seats	NT	University.Research_Staff.relation	SLIM	NO
University.Department	SYN	University.School_Member.faculty	SLIM	YES
University.School_Member.faculty	SYN	University.Department.dept_name	SLIM	YES
University.Department	SYN	University.Department.dept_name	SLIM	YES
University.Article.journal	SYN	Computer_Science.Essay.journal	SLIM	YES
Computer_Science.Publication	BT	University.Article.journal	SLIM	YES
Computer_Science.Publication	BT	Computer_Science.Essay.journal	SLIM	YES
University.Article	NT	Computer_Science.Publication	TRE*	YES

* = Thesaurus Relations Editor.

We recall briefly the source of each type of relations (see sec. 4.1.1 for a complete description):

- **SIMA** is the module that analyze the local sources schemas to find intra- and inter-schema relationships.
- **SIMB** is the module that checks (validates) the correctness of attribute relationships and infers new relationships using ODB-Tools.
- **SLIM** is the module that infers relationships exploiting the semantic annotation of the local terms using the lexical database of WordNet.
- **TRE** is the Thesaurus Relations Editor that guide the user inserting specific relationships into the Thesaurus.

With respect to the annotation of the local terms, we show some examples.

Each term has a name and a meaning; each meaning is composed of a word form (or lemma) and a counter that represents the sense associated with those word form. Some terms can be annotated with more than one meaning. Below each term there is the complete WordNet sense.

```
Source.Class.Attribute = <name, {lemma#sense}>
```

Local terms annotation.

```
Computer_Science.CS_Person = <person,{person#1}>
    person#1 = a human being

Computer_Science.Professor = <professor,{professor#1}>
    professor#1 = someone who is a member of the faculty at a college or university

University.School_Member = <student,{student#1}>
    student#1 = a learner who is enrolled in an educational institution

Computer_Science.Essay = <essay,{essay#1}>
    essay#1 = an analytic or interpretive literary composition

Computer_Science.Publication = <publication,{publication#2}>
    publication#2 = a copy of a printed work offered for distribution

University.Article = <article,{article#1}>
    article#1 = nonfictional prose forming an independent part of a publication

Computer_Science.Course = <course,{course#1}>
    course#1 = education imparted in a series of lessons or class meetings

University.Section = <section,{course#1}>
    course#1 = education imparted in a series of lessons or class meetings

Computer_Science.Office = <office,{office#1}>
    office#1 = place of business where professional or clerical duties are performed

Computer_Science.Student.first_name = <first_name,{first name#1}>
    first name#1 = the name that precedes the surname

Computer_Science.Student.last_name = <last_name,{last name#1}>
    last name#1 = the name used to identify the members of a family

Computer_Science.Professor.name = <name,{name#1}>
    name#1 = a language unit by which a person or thing is known

Computer_Science.Professor.rank = <rank,{rank#2}>
    rank#2 = relative status
```

```
Computer_Science.Student.rank = <rank,{rank#4}>
    rank#4 = position in a social hierarchy

Computer_Science.Professor.title = <title,{title#6,tile#9}>
    title#6 = an identifying appellation signifying status or function
    title#9 = an appellation signifying nobility
```

The next step is the Clusters generation. The integration process produces seven clusters.

Clusters.

Global$_0$ (Course)
- Computer_Science.Course
- University.Section

Global$_1$ (Department)
- University.Department

Global$_2$ (Location)
- Computer_Science.Location

Global$_3$ (Office)
- Computer_Science.Office

Global$_4$ (Person)
- Computer_Science.CS_Person
- Computer_Science.Professor
- University.Research_Staff
- University.School_Member

Global$_5$ (Publication)
- Computer_Science.Essay
- Computer_Science.Publication
- University.Article

Global$_5$ (Publication)
Computer_Science.ClassRoom
University.Room

Figure A.2 is a screenshot of the Clusters generation tab of the MOMIS system; the designer can refine the automatic elaboration of the clusters.

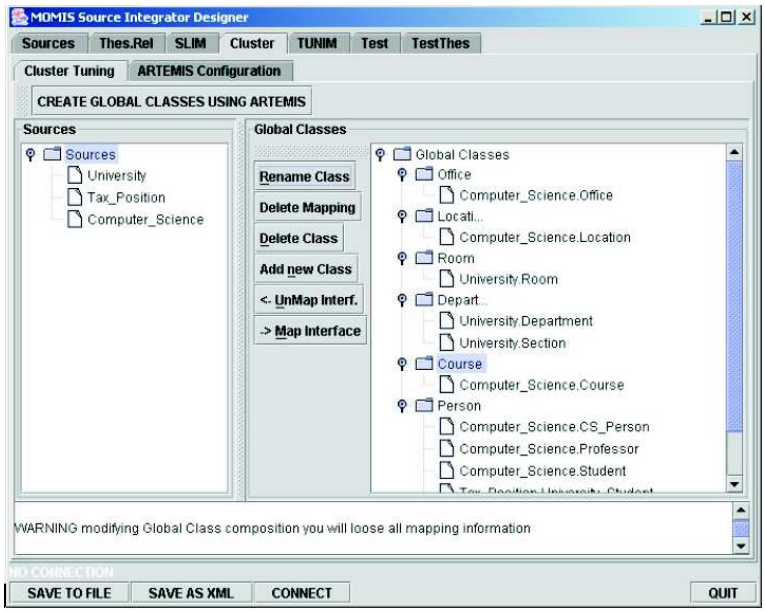

Figure A.2: Cluster generation tab.

After the creation of clusters there is the Global Class generation; for each global class the system propose a mapping table to the designer that can refine the proposed mapping rules. We present here the ODL$_{I^3}$ representation of the global classes.

Global Classes.

```
interface Global0 (Course) {
   attribute string course_name;
      mapping_rules(
         Computer_Science.Course: course_name
         University.Section: section_name
                   )
```

```
   attribute integer length;
      mapping_rules(
         Computer_Science.Course: NULL
         University.Section: length
                  )
   attribute integer room_code;
      mapping_rules(
         Computer_Science.Course: NULL
         University.Section: room_code
                  )
   attribute integer section_code;
      mapping_rules(
         Computer_Science.Course: NULL
         University.Section: section_code
                  )
   attribute complex taught_by;
      mapping_rules(
         Computer_Science.Course: taught_by
         University.Section: NULL
                  )
   attribute complex taught_in;
      mapping_rules(
         Computer_Science.Course: taught_in
         University.Section: NULL
                  )
}

interface Global1 (Department) {
   attribute integer budget;
      mapping_rules(
         University.Department: budget
                  )
   attribute integer dept_code;
      mapping_rules(
         University.Department: dept_code
                  )
   attribute string dept_name;
      mapping_rules(
         University.Department: dept_name
                  )
}
```

```
interface Global2 (Location) {
   attribute string building_name;
      mapping_rules(
         Computer_Science.Location: building_name
                  )
   attribute integer floor;
      mapping_rules(
         Computer_Science.Location: floor
                  )
   attribute integer number;
      mapping_rules(
         Computer_Science.Location: number
                  )
}

interface Global3 (Office) {
   attribute complex address;
      mapping_rules(
         Computer_Science.Office: address
                  )
   attribute string code;
      mapping_rules(
         Computer_Science.Office: code
                  )
   attribute string description;
      mapping_rules(
         Computer_Science.Office: description
                  )
}

interface Global4 (Person) {
   attribute complex belongs_to;
      mapping_rules(
         Computer_Science.CS_Person: NULL
         Computer_Science.Professor: belongs_to
         Computer_Science.Student: NULL
         University.Research_Staff: NULL
         University.School_Member: NULL
                  )
   attribute integer dept_code;
      mapping_rules(
         Computer_Science.CS_Person: NULL
```

```
      Computer_Science.Professor: NULL
      Computer_Science.Student: NULL
      University.Research_Staff: dept_code
      University.School_Member: NULL
               )
attribute string e_mail;
   mapping_rules(
      Computer_Science.CS_Person: NULL
      Computer_Science.Professor: NULL
      Computer_Science.Student: NULL
      University.Research_Staff: e_mail
      University.School_Member: NULL
               )
attribute string faculty;
   mapping_rules(
      Computer_Science.CS_Person: NULL
      Computer_Science.Professor: NULL
      Computer_Science.Student: NULL
      University.Research_Staff: NULL
      University.School_Member: faculty
               )
attribute string name;
   mapping_rules(
      Computer_Science.CS_Person: first_name AND last_name
      Computer_Science.Professor: first_name AND last_name
      Computer_Science.Student: first_name AND last_name
      University.Research_Staff: name
      University.School_Member: name
               )
attribute string rank;
   mapping_rules(
      Computer_Science.CS_Person: NULL
      Computer_Science.Professor: rank
      Computer_Science.Student: rank
      University.Research_Staff: NULL
      University.School_Member: NULL
               )
attribute string relation;
   mapping_rules(
      Computer_Science.CS_Person: NULL
      Computer_Science.Professor: NULL
      Computer_Science.Student: NULL
      University.Research_Staff: relation
```

```
         University.School_Member: NULL
                  )
   attribute integer section_code;
      mapping_rules(
         Computer_Science.CS_Person: NULL
         Computer_Science.Professor: NULL
         Computer_Science.Student: NULL
         University.Research_Staff: section_code
         University.School_Member: NULL
                  )
   attribute array takes;
      mapping_rules(
         Computer_Science.CS_Person: NULL
         Computer_Science.Professor: NULL
         Computer_Science.Student: takes
         University.Research_Staff: NULL
         University.School_Member: NULL
                  )
   attribute string title;
      mapping_rules(
         Computer_Science.CS_Person: NULL
         Computer_Science.Professor: title
         Computer_Science.Student: NULL
         University.Research_Staff: NULL
         University.School_Member: NULL
                  )
   attribute integer year;
      mapping_rules(
         Computer_Science.CS_Person: NULL
         Computer_Science.Professor: NULL
         Computer_Science.Student: year
         University.Research_Staff: NULL
         University.School_Member: year
                  )
}

interface Global5 (Publication) {
   attribute integer article_code;
      mapping_rules(
         Computer_Science.Essay: NULL
         Computer_Science.Publication: NULL
         University.Article: article_code
                  )
```

```
   attribute array authors;
      mapping_rules(
         Computer_Science.Essay: authors
         Computer_Science.Publication: authors
         University.Article: author
                  )
   attribute string journal;
      mapping_rules(
         Computer_Science.Essay: journal
         Computer_Science.Publication: NULL
         University.Article: journal
                  )
   attribute string subject;
      mapping_rules(
         Computer_Science.Essay: subject
         Computer_Science.Publication: NULL
         University.Article: subject
                  )
   attribute string title;
      mapping_rules(
         Computer_Science.Essay: title
         Computer_Science.Publication: title
         University.Article: title
                  )
   attribute integer year;
      mapping_rules(
         Computer_Science.Essay: year
         Computer_Science.Publication: year
         University.Article: year
                  )
}

interface Global6 (Room) {
   attribute complex address;
      mapping_rules(
         Computer_Science.ClassRoom: address
         University.Room: NULL
                  )
   attribute integer code;
      mapping_rules(
         Computer_Science.ClassRoom: code
         University.Room: room_code
                  )
```

```
   attribute string notes;
      mapping_rules(
         Computer_Science.ClassRoom: notes
         University.Room: notes
                   )
   attribute integer seats;
      mapping_rules(
         Computer_Science.ClassRoom: seats
         University.Room: seats_number
                   )
}
```

Each Global Class can be shown as a table where each row of the first column represents the global attributes while the others columns represent the local classes that belong to the global class. Each element of the table represent the local attributes mapped by the corresponding global attribute. We show also a screenshot of the MOMIS tab that help the designer in the refinement of the mapping tables.

Global Class **Global**$_4$ (**Person**)

	CS	CS	CS	UNI	UNI
	CS_Person	Professor	Student	Research_Staff	School_Member
belongs_to	null	belongs_to	null	null	null
dept_code	null	null	null	dept_code	null
e_mail	null	null	null	e_mail	null
faculty	null	null	null	null	faculty
name	first_name and last_name	first_name and last_name	first_name and last_name	name	name
rank	null	rank	rank	null	null
relation	null	null	null	relation	null
section_code	null	null	null	section_code	null
title	null	title	null	null	null
year	null	null	year	null	year

Figure A.3 shows the SI-Designer's tab where the mapping tables of the global attributes are refined by the designer.

Following the proposal that we have formulated for the integration of new sources, we have first to annotate the terms (classes and attributes) of the Global Schema (GVV). Doing this we respect the rules presented in section 5.4.1. Some results are here provided.

Global terms annotation.

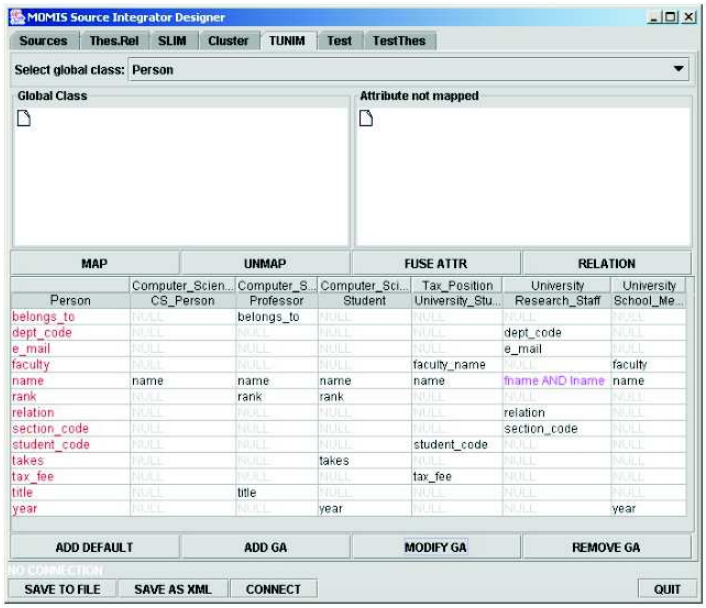

Figure A.3: Global Attributes refinement.

```
Global0 = <course,{course#1}>
        course#1 = education imparted in a series of lessons or class meetings

Global3 = <office,{office#1}>
    office#1 = place of business where professional or clerical duties are performed


Global4 = <person,{person#1,professor#1,student#1}>
    person#1 = a human being
    professor#1 = someone who is a member of the faculty at a college or university
    student#1 = a learner who is enrolled in an educational institution

Global5 = <publication,{essay#1,publication#2,article#1}>
    essay#1= an analytic or interpretive literary composition
    publication#2 = a copy of a printed work offered for distribution
    article#1 =nonfictional prose forming an independent part of a publication

Global4.title = <title,{title#6,tile#9}>
    title#6 = an identifying appellation signifying status or function
    title#9 = an appellation signifying nobility

Global4.rank = <rank,{rank#2,rank#4}>
    rank#2 = relative status
    rank#4 = position in a social hierarchy
```

```
Global4.name = <name,{first name#1,last name#1,name#1}>
    first name#1 = the name that precedes the surname
    last name#1 = the name used to identify the members of a family
    name#1 = a language unit by which a person or thing is known
```

Now we present the XML schema of the new source **MATH** that we try to integrate into the previous GVV.

The new source describes the information related to persons, students, courses, examinations, and papers of a math department.

Source **Math**

```
<Source name="Math" type="xml">
        <Interface name="Teacher" persistent="false">
           <extends name="Person"/>
           <extent name="Teachers"/>
           <Attribute name="rank" type="string"/>
           <Attribute name="teach" type="set" collectionOf="Course"/>
           <Attribute name="write" type="set" collectionOf="Paper"/>
           <Attribute name="work_in" type="Office"/>
        </Interface>

        <Interface name="Office" persistent="false">
           <extent name="Offices"/>
           <Key name="PrimaryKey">
             <KeyAttribute name="code"/>
           </Key>
           <Attribute name="code" type="string"/>
           <Attribute name="section" type="string"/>
           <Attribute name="floor" type="long"/>
        </Interface>

        <Interface name="Student" persistent="false">
           <extends name="Person"/>
           <extent name="Students"/>
           <Attribute name="year" type="long"/>
           <Attribute name="attend" type="set" collectionOf="Course"/>
        </Interface>

        <Interface name="Course" persistent="false">
           <extent name="Courses"/>
           <Attribute name="course_name" type="string"/>
```

```
      <Attribute name="taught_in" type="ClassRoom"/>
      <Attribute name="period" type="string"/>
    </Interface>

    <Interface name="ClassRoom" persistent="false">
      <extent name="ClassRooms"/>
      <Attribute name="code" type="string"/>
      <Attribute name="seats" type="long"/>
      <Attribute name="description" type="string"/>
    </Interface>

    <Interface name="Person" persistent="false">
      <extent name="Persons"/>
      <Key name="PrimaryKey">
        <KeyAttribute name="first_name"/>
        <KeyAttribute name="last_name"/>
      </Key>
      <Attribute name="first_name" type="string"/>
      <Attribute name="last_name" type="string"/>
      <Attribute name="birth_date" type="string"/>
    </Interface>

    <Interface name="Researcher" persistent="false">
      <extends name="Person"/>
      <extent name="Researcher"/>
      <Attribute name="write" type="set" collectionOf="Paper"/>
      <Attribute name="work_in" type="Office"/>
      <Attribute name="subject" type="string"/>
    </Interface>

  <Interface name="Examination" persistent="false">
      <extent name="Examinations"/>
      <Key name="PrimaryKey">
        <KeyAttribute name="candidate"/>
        <KeyAttribute name="course"/>
      </Key>
      <Attribute name="candidate" type="Student"/>
      <Attribute name="course" type="Course"/>
      <Attribute name="result" type="long"/>
    </Interface>

  <Interface name="Paper" persistent="false">
      <extent name="Papers"/>
```

```
            <Key name="PrimaryKey">
              <KeyAttribute name="title"/>
            </Key>
            <Attribute name="title" type="string"/>
            <Attribute name="year" type="long"/>
            <Attribute name="journal" type="string"/>
            <Attribute name="subject" type="string"/>
        </Interface>
</Source>
```

The integration process with the previous GVV produces the following clusters.

Clusters.

NewGlobal$_0$ (Location)
GVV.Location

NewGlobal$_1$ (Department)
GVV.Department

NewGlobal$_2$ (Room)
GVV.Room
Math.ClassRoom

NewGlobal$_3$ (Office)
GVV.Office
Math.Office

NewGlobal$_4$ (Examination)
Math.Examination

NewGlobal$_5$ (Person)
- GVV.Person
- Math.Person
- Math.Researcher
- Math.Student
- Math.Teacher

NewGlobal$_6$ (Publication)
- GVV.Publication
- Math.Paper

Appendix B

The ODL$_{I^3}$ description language

We present here the BNF description of ODL$_{I^3}$ language. We illustrate only the syntactic elements that differ from the original ODL grammar defined in ODMG-93 standard.

⟨interface_dcl⟩ ::= ⟨interface_header⟩
{[⟨ interface_body⟩]**}**;
[**union** ⟨identifier⟩ **{** ⟨interface_body⟩ **};**]

⟨interface_header⟩ ::= **interface** ⟨identifier⟩
[⟨inheritance_spec⟩]
[⟨type_property_list⟩]

⟨inheritance_spec⟩ ::= : ⟨scoped_name⟩
[,⟨inheritance_spec⟩]

Definition of local schema model: wrappers have to be able to specify source's type and name for each model.

⟨type_property_list⟩ ::= **(** [⟨source_spec⟩]
[⟨extent_spec⟩]
[⟨key_spec⟩] [⟨f_key_spec⟩] [⟨c_key_spec⟩] **)**
⟨source_spec⟩ ::= **source** ⟨source_type⟩
⟨source_name⟩
⟨source_type⟩ ::= **relational** | **nfrelational**
| **object** | **file**
| **semistructured**
⟨source_name⟩ ::= ⟨identifier⟩
⟨extent_spec⟩ ::= **extent** ⟨extent_list⟩
⟨extent_list⟩ ::= ⟨string⟩ | ⟨string⟩,⟨extent_list⟩
⟨key_spec⟩ ::= **key**[**s**] ⟨key_list⟩
⟨f_key_spec⟩ ::= **foreign_key** (⟨f_key_list⟩)
references ⟨key_list
⟩ [⟨f_key_spec⟩]
⟨c_key_spec⟩ ::= **candidate_key** ⟨identifier⟩
(⟨key_list⟩**)**

Definition of mapping rules between global attributes and corresponding local attributes of local sources.

⟨attr_dcl⟩ ::= [**readonly**] **attribute**
[⟨domain_type⟩]
⟨attribute_name⟩ [*]
[⟨fixed_array_size⟩]
[⟨mapping_rule_dcl⟩]
⟨mapping_rule_dcl⟩ ::= **mapping_rule** ⟨rule_list⟩
⟨rule_list⟩ ::= ⟨rule⟩ | ⟨rule⟩,⟨rule_list⟩
⟨rule⟩ ::= ⟨local_attr_name⟩ |
'⟨identifier⟩'
⟨and_expression⟩ |
⟨union_expression⟩
⟨and_expression⟩ ::= **(** ⟨local_attr_name⟩ **and**
⟨and_list⟩ **)**
⟨and_list⟩ ::= ⟨local_attr_name⟩
| ⟨local_attr_name⟩ **and**
⟨and_list⟩
⟨union_expression⟩ ::= **(** ⟨local_attr_name⟩ **union**
⟨union_list⟩ **on** ⟨identifier⟩ **)**
⟨union_list⟩ ::= ⟨local_attr_name⟩
| ⟨local_attr_name⟩ **union**
⟨union_list⟩
⟨local_attr_name⟩ ::= ⟨source_name⟩.⟨class_name⟩.
⟨attribute_name⟩
...

Terms relationships inserted in the Common Thesaurus.

⟨relationships_list⟩ ::= ⟨relationship_dcl⟩**;** |
⟨relationship_dcl⟩**;**
⟨relationships_list⟩
⟨relationships_dcl⟩ ::= ⟨local_name⟩
⟨relationship_type⟩
⟨local_name⟩
⟨local_name⟩ ::= ⟨source_name⟩.
⟨local_class_name⟩
[.⟨local_attr_name⟩]
⟨relationship_type⟩ ::= **SYN** | **BT** | **NT** | **RT**
...

Definition of **OLCD** integrity constraints: rule declarations (using *if then* definition) valid for each data instance; mapping rule declarations (specification of *or*, *and* rules).

⟨rule_list⟩ ::= ⟨rule_dcl⟩**;** | ⟨rule_dcl⟩**;** ⟨rule_list⟩
⟨rule_dcl⟩ ::= **rule** ⟨identifier⟩ ⟨rule_spec⟩
⟨rule_spec⟩ ::= ⟨rule_pre⟩ **then** ⟨rule_post⟩ |
{ ⟨case_dcl⟩ **}**
⟨rule_pre⟩ ::= ⟨forall⟩ ⟨identifier⟩ **in** ⟨identifier⟩ **:**
⟨rule_body_list⟩
⟨rule_post⟩ ::= ⟨rule_body_list⟩
⟨case_dcl⟩ ::= **case of** ⟨identifier⟩ **:** ⟨case_list⟩
⟨case_list⟩ ::= ⟨case_spec⟩ | ⟨case_spec⟩ ⟨case_list⟩
⟨case_spec⟩ ::= ⟨identifier⟩ **:** ⟨identifier⟩ **;**
⟨rule_body_list⟩ ::= **(** ⟨rule_body_list⟩ **)** |
⟨rule_body⟩ |
⟨rule_body_list⟩ **and**
⟨rule_body⟩ |
⟨rule_body_list⟩ **and**
(⟨rule_body_list⟩ **)**
⟨rule_body⟩ ::= ⟨dotted_name⟩
⟨rule_const_op⟩
⟨literal_value⟩ |
⟨dotted_name⟩
⟨rule_const_op⟩
⟨rule_cast⟩ ⟨literal_value⟩ |
⟨dotted_name⟩ **in**
⟨dotted_name⟩ |
⟨forall⟩ ⟨identifier⟩ **in**
⟨dotted_name⟩ **:**
⟨rule_body_list⟩ |
exists ⟨identifier⟩ **in**
⟨dotted_name⟩ **:**
⟨rule_body_list⟩
⟨rule_const_op⟩ ::= = | ≥ | ≤ | > | <
⟨rule_cast⟩ ::= **(**⟨simple_type_spec⟩**)**
⟨dotted_name⟩ ::= ⟨identifier⟩ | ⟨identifier⟩**.**
⟨dotted_name⟩
⟨forall⟩ ::= **for all** | **forall**

Appendix C

Links

Chapter 1 – The Semantic Web

Scientific American
http://www.sciam.com/

Agent Link
http://www.agentlink.org/

Chapter 2 – SEWASIE and MOMIS

SEWASIE
http://www.sewasie.org/

MOMIS
http://www.dbgroup.unimo.it/Momis/

Object Management Group
http://www.omg.org/

Chapter 3 – Ontologies

Gruber's definition of ontology
http://www-ksl.stanford.edu/kst/what-is-an-ontology.html

Chapter 4 – Ontology management -tools and projects

OILEd
http://oiled.man.ac.uk/

OntoEdit
http://www.ontoprise.de/ontoedit_en.htm

Protégé-2000 and **PROMPT**
http://protege.stanford.edu/

WebODE
http://delicias.dia.fi.upm.es/webODE/

Chimaera
http://www.ksl.stanford.edu/software/chimaera/

Chapter 6 – Semantic Web projects

World Wide Web Consortium W3C
http://www.w3c.org/
Semantic Web Working Group
http://www.w3c.org/sw/

OWL guide
http://www.w3.org/TR/owl-guide/
OWL technical reference
http://www.w3.org/TR/owl-ref/

OntoWeb
http://www.ontoweb.org/
http://babage.dia.fi.upm.es/ontoweb/wp1/OntoRoadMap/index.html
OntoWeb deliverables
http://ontoweb.aifb.uni-karlsruhe.de/About/Deliverables

On-To-Knowledge
http://www.ontoknowledge.org/

BT Exact Technologies
http://www.btexact.com/

AIdministrator
http://www.aidministrator.nl/

OntoEdit
http://www.ontoprise.de/ontoedit.htm

OntoText Lab
http://www.ontotext.com/

Ontology Middleware Module
http://www.ontotext.com/omm

Sesame
http://sesame.aidministrator.nl
http://sourceforge.net/projects/sesame/

BOR
http://www.ontotext.com/bor

CognIt
http://www.cognit.no/

CORPORUM
http://ontoserver.cognit.no/

International Semantic Web Conference (ISWC)
http://iswc.semanticweb.org/

DAML
http://www.daml.org/

KAON
http://kaon.semanticweb.org

Simple HTML Ontology Extension (SHOE)
http://www.cs.umd.edu/projects/plus/SHOE/index.html

SHOE specifications
http://www.cs.umd.edu/projects/plus/SHOE/spec.html

MIND SWAP
http://www.mindswap.org/

Bibliography

[1] J. Hendler, O. Lassila, and T. Berners-Lee. The semantic web, a new form of web content that is meaningful to computers will unleash a revolution of new possibilities. *Scientific American*, May 2001.

[2] T. R. Gruber. A translation approach to portable ontology specifications. In *Knowledge Acquisition*, volume 5, pages 199–220, 1993.

[3] M. Wooldridge and N.R. Jennings. Intelligent agents: Theories and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.

[4] S. Bergamaschi, S. Castano, and M. Vincini. Semantic integration of semistructured and structured data sources. *SIGMOD Records*, 28(1), March 1999.

[5] Domenico Beneventano, Sonia Bergamaschi, Silvana Castano, Alberto Corni, R. Guidetti, G. Malvezzi, Michele Melchiori, and Maurizio Vincini. Information integration: The momis project demonstration. In Amr El Abbadi, Michael L. Brodie, Sharma Chakravarthy, Umeshwar Dayal, Nabil Kamel, Gunter Schlageter, and Kyu-Young Whang, editors, *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt*, pages 611–614. Morgan Kaufmann, 2000.

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

[7] I. Benetti, D. Beneventano, S. Bergamaschi, F. Guerra, and M. Vincini. An information integration framework for e-commerce. *IEEE Intelligent Systems Magazine*, Jannuary/February 2002.

[8] D. Beneventano, S. Bergamaschi, C. Sartori, and M. Vincini. Odb-tools: a description logics based tool for schema validation and semantic query

optimization in object oriented databases. In *Sesto Convegno AIIA - Roma*, 1997.

[9] A.G. Miller. Wordnet: A lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[10] Castano S., De Antonellis V., and De Capitani Di Vimercati S. Global viewing of heterogeneous data sources. *IEEE Transactions TKDE*, (13(2)):277–297, 2001.

[11] R. Hull and R. King et al. Arpa i^3 reference architecture, 1995. Available at http://www.isse.gmu.edu/I3_Arch/index.html.

[12] V. Tamma. *An Ontology Model supporting Multiple Ontologies for Knowledge sharing.* PhD thesis, Department of Computer Science, University of Liverpool, 2001.

[13] N. Guarino. Formal ontologies and information systems. In *Proceedings of the International Conference on Formal Ontology in Information Systems (FOIS'98)*, Trento, Italy, june 1998.

[14] P. Borst. *Construction of Engineering ontologies for knowledge sharing and reuse.* PhD thesis, University of Twente, Centre for Telematica and Information Technology, 1997.

[15] R. Studer, V.R. Benjamins, and D. Fensel. Knowledge engineering, principles and methods. In *Data and Knowledge Engineering*, volume 25, pages 161–197, 1998.

[16] R. Neches, R.E. Fikes, T. Finin, T.R. Gruber, R. Patil, and T. Senator. Enabling technology for knowledge sharing. *AI Magazine*, 12(3):36–56, 1991.

[17] A. Bernaras, I. Laresgoiti, and J. Corera. Building and reusing ontologies for electrical network applications. In W. Wahlster, editor, *Proceedings of the 12th European Conference on Artificial Intelligence (ECAI)*, pages 298–302, Chichester, England, 1996. John Wiley Sons, Ltd.

[18] B. Swartout, R. Patil, K. Knight, and T. Russ. Towards distributed use of largescale ontologies. In University of Calgary, editor, *In Proceedings of Tenth Knowledge Acquisition for Knowledge-Based Systems Workshop (KAW)*, Banff, Alberta, Canada, 1996.

[19] M. Uschold. Knowledge level modelling: concepts and terminology. *Knowledge Engineering Review*, 1998.

[20] G. van Heijst, A.Th. Schreiber, and B.J. Wielinga. Using explicit ontologies in kbs development. *International Journal of Human-Computer Studies*, 45:184–292, 1997.

[21] M. Uschold and M. Gruninger. Ontologies: principles, methods, and applications. *Knowledge Engineering Review*, 11(2):93–155, 1996.

[22] A. Farquhar, R. Fikes, and J. Rice. The ontolingua server: a tool for collaborative ontology construction. *International Journal of Human-Computer Studies*, 46:707–728, 1997.

[23] O. Lassila and D. McGuinness. The role of frame-based representation on the semantic web. *Electronic Transactions on Artificial Intelligence (ETAI) Journal: area The Semantic Web*, To appear, 2001.

[24] M. Klein and D. Fensel. Ontology versioning on the semantic web. In *International Semantic Web Working Symposium (SWWS)*, pages 75–91, Stanford University, USA, July 30 - August 1 2001.

[25] Ljiljana Stojanovic and Boris Motik. Ontology evolution within ontology editors. In *2002 in 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW2002)*, Siguenza, Spain, October 1-4 2002.

[26] L. Stojanovic, A. Maedche, B. Motik, and N. Stojanovic. User-driven ontology evolution management. In *13th International Conference on Knowledge Engineering and Knowledge Management (EKAW2002)*, Siguenza, Spain, October 1-4 2002.

[27] N. F. Noy and D. McGuinness. Ontology development 101: A guide to creating your first ontology. Technical Report KLS-01-05, Standford KSL, 2000.

[28] Y. Sure. On-to-knowledge – ontology based knowledge management tools and their application. *German Journal Kuenstliche Intelligenz, Special Issue on Knowledge Management*, (1/02), 2002.

[29] S. Castano and V. De Antonellis. Deriving global conceptual views from multiple information sources. In *preProc. of ER'97 Preconference Symposium on Conceptual Modeling, Historical Perspectives and Future Directions*, 1997.

[30] Veronica Guidetti. Intelligent information integration systems: extending lexicon ontology. Master's thesis, Università di Modena

e Reggio Emilia, Facoltà di Ingegneria, corso di laurea in Ingegneria Informatica, Modena, Italy, 2001-2002. Available from http://www.dbgroup.ing.unimo.it/.

[31] Asunción Gómez Pérez. A survey on ontology tools. Deliverable 1.3, OntoWeb Ontology-based information exchange for knowledge management and electronic commerce, IST-2000-29243, May 2002.

[32] Natalya F. Noy and Mark A. Musen. Evaluating ontology-mapping tools: Requirements and experience. In *2002 in 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW2002)*, Siguenza, Spain, October 1-4 2002.

[33] A. Maedche, B. Motik, N. Silva, and R. Volz. Mafra-a mapping framework for distributed ontologies. In *13th International Conference on Knowledge Engineering and Knowledge Management (EKAW2002)*, Siguenza, Spain, October 1-4 2002.

[34] J. Heflin and J. Hendler. Dynamic ontologies on the web. In AAAI/MIT Press, editor, *7th National Conference on Artificial Intelligence (AAAI-2000)*, pages 443–449, Menlo Park, CA, USA, 2000.

[35] Lorenzo Lugli. Integrazione di sorgenti html in momis: Analisi comparativa degli strumenti esistenti. Master's thesis, Università di Modena e Reggio Emilia, Facoltà di Ingegneria, corso di laurea in Ingegneria Informatica, Modena, Italy, 2001-2002. Available from http://www.dbgroup.ing.unimo.it/.