Deliverable 1.4: A survey on methodologies for developing, maintaining, evaluating and reengineering ontologies

Identifier	Deliverable 1.4
Class	**Deliverable**
Version	**1.0**
Version date	**12/12/2002**
Status	**Final**
Distribution	**Public**
Responsible Partner	**UPM**

OntoWeb: Ontology-based Information Exchange for Knowledge Management and Electronic Commerce

OntoWeb: Ontology-based Information Exchange for Knowledge Management and Electronic Commerce

OntoWeb Consortium

This document is part of a research project funded by the IST Programme of the Commission of the European Communities as project number IST-2000-29243.

Vrije Universiteit Amsterdam (VU)-Coordinator
Faculty of Sciences,
Division of Mathematics and Computer Science
De Boelelaan 1081a
1081 HV Amsterdam, the Netherlands
Fax and Answering machine: +31-(0)84-872 27 22
Mobil phone: +31-(0)6-51850619
Contactperson: Dieter Fensel
E-mail: dieter@cs.vu.nl

This document has been edited by:

Mariano Fernández-López
Departamento de Inteligencia Artificial
Facultad de Informática (Universidad Politécnica de Madrid)
Campus de Montegancedo s/n. 28660 Boadilla del Monte. Madrid. Spain

The main contributors of this document are:

Mariano Fernández-López (UPM)

Asunción Gómez-Pérez (UPM)

Additional contributors for each section are presented below in alphabetical order:

Chapter 2: York Sure (University of Karlsruhe) who provided On-To-Knowledge description and comparison values for this methodology.

Chapter 4: Jérôme Euzenat (INRIA), who provided Co4 description.

Chapter 5: Aldo Gangemi (ISTC-CNR), Domenico M. Pisanelli (ISTC-CNR) and Gerardo Steve (ISTC-CNR), who provided ONIONS description.

Gerd Stumme (University of Karlsruhe), who provided FCA-Merge description.

Yannis Kalfoglou (AKT) and Marco Schorlemmer (AKT) who provided Information-Flow-based Ontology Mapping description.

Sonia Bergamaschi, Domenico Beneventano and Francesco Guerra and Maurizio Vincini (Dipartimento di Ingegneria dell'Informazione Università di Modena e Reggio Emilia), who provided MOMIS descriptions.

Chapter 6: Ljiljana Stojanovic (German Research Center FZI) who provided the ontology evolution study.

Chapter 7: Yannis Kalfoglou (AKT), who provided Ontological Constraints Manager description.

Revision Information

Revision date	Version	Changes
14-06-2002	0.1	Table of Content proposal
08-07-2002	0.15	UPM describes methodologies for building ontologies from scratch
23-07-2002	0.2	UPM describes methodologies for reengineering ontologies
08-08-2002	0.25	UPM describes methodologies for cooperative construction of ontologies
14-08-2002	0.3	UPM includes list ontology merging methodologies
03-09-2002	0.35	UPM describes ontology evaluation methods
17-09-2002	0.4	UPM distributes first deliverable version
18-09-2002	0.45	University of Karlsruhe sends FCA-Merge description
19-09-2002	0.5	AKT sends description of OCM
24-09-2002	0.55	INRIA sends Co4 description
25-09-2002	0.6	AKT sends Information-Flow-based Ontology Mapping description
15-10-2002	0.66	ISTC-CNR sends ONIONS description
18-10-2002	0.7	UPM prepares the first broadcast version
24-10-2002	0.75	University of Karlsruhe sends On-To-Knowledge description, On-To-Knowledge comparison values and ontology evolution proposal
08-11-2002	0.8	UPM prepares de second broadcast version
08-11-2002	0.85	Modena Universty sends MOMIS description
25-11-2002	0.9	ISTC-CNR updates ONIONS description
12-12-2002	1.0	Delivery to the Project Coordinator

Contents

OntoWeb Consortium....3
Revision Information....4
Contents....6
Executive summary....9
1 Introduction....10
2 Methodologies and methods for building ontologies from scratch....11
2.1 Introduction....11
2.2 Evaluation framework for methodologies and methods....11
2.3 Description of methodologies and methods for building ontologies from scratch....12
2.3.1 Cyc method....12
2.3.2 Uschold and King's method....13
2.3.3 Grüninger and Fox's methodology....14
2.3.4 KACTUS method....16
2.3.5 METHONTOLOGY....17
2.3.6 SENSUS method....20
2.3.7 On-To-Knowledge Methodology....21
2.4 Comparison of approaches against the evaluation framework....24
3 Method for reengineering ontologies....29
3.1 Method for reengineering ontologies of the Ontology Group at UPM....29
4 Methods for cooperative construction of ontologies....32
4.1 Introduction....32
4.2 Description of methods for cooperative construction of ontologies....32
4.2.1 Co_4: Collaborative construction of consensual knowledge bases....32
4.2.2 $(KA)^2$ method....33
5 Ontology merging methods and methodologies....35
5.1 Introduction....35
5.2 Description of methods and methodologies for ontology merging....35
5.2.1 ONIONS....35
5.2.2 PROMPT....38
5.2.3 FCA-Merge – A Method for Bottom-Up Merging of Ontologies....40
5.2.4 Information-Flow-based Ontology Mapping....41
5.2.5 The MOMIS methodology....41
6 Ontology evolution methods....44
7 Ontology evaluation methods....47
7.1 Introduction....47
7.2 Description of methods for ontology evaluation....47
7.2.1 Gómez Pérez's approach for taxonomy evaluation....47
7.2.2 Ontological Constraints Manager (OCM)....50

7.2.3 OntoClean....53
8 Conclusions....55

Executive summary

This deliverable presents a survey of the most relevant methodologies and methods used for building, maintaining, evaluating and reengineering ontologies. This survey is divided into several sections, which group different kinds of approaches, namely:

- Methodologies and methods for building ontologies from scratch. In particular the approaches to be shown will be the Cyc method, Uschold and King's method, Grüninger and Fox's methodology, KACTUS method, METHONTOLOGY, SENSUS method, and the On-to-Knowledge methodology.
- A method for reengineering ontologies.
- Methods for cooperative construction of ontologies: Co4 and $(KA)^2$
- Ontology merging methods. We have included: ONIONS method, PROMPT, FCA-Merge, IFOM and MOMIS.
- Ontology evolution methods.
- Ontology evaluation methods. We have included: Gómez-Pérez's approach for taxonomy evaluation, OCM and OntoClean.

Whenever it is possible, for each approach, we will present: its general description, the URL where you can find information about it, its bibliographic references, its recommended life cycle, the general steps proposed by the approach, the tools that provide technological support to it, and significant ontologies built following it.

This survey has been performed in conjunction with partners of the Ontoweb workpackage 1 and other participants that have elaborated on their specific approaches.

1 Introduction

A methodology is a "comprehensive, integrated series of techniques or methods creating a general systems theory of how a class of thought-intensive work ought be performed" [IEEE, 95]. A method is an 'orderly' process or procedure used in the engineering of a product or performing a service [IEEE, 90]. A technique is a "technical and managerial procedure used to achieve a given objective" [IEEE, 95]. Hence, the main difference between methods and techniques is that a method requires an order and a technique does not. Both, methods and techniques, are parts of methodologies.

A number of approaches have been reported for developing ontologies. Some of them describe how to build an ontology from scratch or reusing other ontologies (Cyc approach [Lenat et al.; 90], Uschold and King's proposal [Uschold et al.; 95], METHONTOLOGY [Fernández López et al.; 99], On-To-Knowledge [Staab et al.; 01], etc.). Other approaches describe how to build an ontology by means of the transformation of other ontologies, for example, using reengineering or ontology merging. There are also proposals for particular activities inside ontology development (evaluation, merging, and evolution, for example).

In this document, we describe different methodologies and methods involved in the different activities of the ontology development process. Section 2 will present methodologies to build ontologies from scratch or reusing other ontologies without transforming them. Section 3 will present an ontology reengineering method, which can be used to reuse existing ontologies. Section 4 will present methods for collaborative ontology construction. Section 5 will present ontology merging methods. Section 6 will present an ontology evolution method. Section 7 will present ontology evaluation methods. Finally, section 8 contains the conclusions of the deliverable.

References

- M. Fernández-López, A. Gómez-Pérez, A. Pazos-Sierra, J. Pazos-Sierra, Building a Chemical Ontology Using METHONTOLOGY and the Ontology Design Environment. IEEE Intelligent Systems & their applications. (January/February 1999) 37-46.
- M. Fernández-López, A. Gómez-Pérez, M.D. Rojas-Amaya, Ontologies' crossed life cycles. Workshop on Knowledge Acquisition, Modelling and Management (EKAW). (Springer Verlag, Jean Les Pins, France 2000). 65-79.
- IEEE 1995. IEEE Guide for Software Quality Assurance Planning. Std. 730.1-1995. IEEE Computer Society. New York (USA).
- IEEE 1990. IEEE Standard Glossary of Software Engineering Terminology. Std. 610.12-1990. IEEE Computer Society. New York (USA).
- D.B. Lenat, R.V. Guha, Building large knowledge-based systems. (Addison-Wesley Publising Company, Inc. 1990).
- S. Staab, H.-P. Schnurr, R. Studer, and Y. Sure: Knowledge Processes and Ontologies, IEEE Intelligent Systems, 16(1), January/February 2001.
- M. Uschold, M. King, Towards a Methodology for Building Ontologies. Workshop on Basic Ontological Issues in Knowledge Sharing. (1995).

2 Methodologies and methods for building ontologies from scratch

2.1 Introduction

The methodologies and methods presented in this section are thought to develop ontologies from scratch or reusing other ontologies as they are available in ontology libraries.

We will first present in section 2.2, the main set of criteria used to compare different approaches of this type. In section 2.3, we will present in depth some of the methodologies and methods that were already included in OntoWeb deliverable D1.1. Section 2.4 compares these approaches with respect to the evaluation framework of section 2.2.

2.2 Evaluation framework for methodologies and methods

The characteristics that we have considered for each approach are classified in the following groups:

1) *Proposed construction strategy*. Here, we include:

   1.1) *Life cycle proposal*: the life cycle identifies the *set of stages* through which the ontology moves during its life time, describes what activities are to be performed in each stage and how the stages are related (relation of precedence, return, etc.).

   1.2) *Strategy with respect the application*. It has to do with the degree of dependency with respect to the application that uses the ontology, the methodologies and methods can be classified into: *application dependent*, which approaches the ontology construction on the basis of an application knowledge base, by means of a process of abstraction; *application semidependent*, which identify, in the specification stage, possible scenarios of ontology use; and *application-independent*, where the process is totally independent of the uses to which the ontology will be put in knowledge-based systems, agents, etc.

   1.3) *Use of core ontologies*. It is possible to use a core ontology as a starting point in the development of the domain ontology.

   1.4) *Strategy to identify concepts* Three possible strategies for identifying concepts: from the most concrete to the most abstract (bottom-up), from the most abstract to the most concrete (top-down), or from the most relevant to the most abstract and most concrete (middle-out).

2) *Proposed ontology development process*. It is based on the framework established in [Fernández-López et al.; 2002], which adapt the IEEE 1075-1995 standard for software development process [IEEE, 1996]. Such process is broken down in other processes (management processes, development-oriented processes, etc.). Processes are made by activities. For each activity of the framework, we set up whether it is proposed or not, and if it is described in detail. The following kinds of processes are distinguished:

   2.1) *Project management processes*. They create the framework for the project and ensure the right level of management throughout the entire product life cycle. Activities related to project initiation (participants, scheduling, etc.), project monitoring and control, and ontology quality management belong to this group of processes.

   2.2) *Ontology development-oriented processes*. Produce, install, operate and maintain the ontology and retire it from use. They are divided into three groups:

   - *Pre-development processes*. They are performed before starting real ontology development. They involve activities related to the study of the ontology installation environment, and to feasibility studies.

   - *Development processes*: They are processes that must be performed to build the ontology. These processes include: *requirements process*, which includes iterative activities that are directed towards developing the ontology requirements specification; *design process*, the goal of which is to develop a coherent and well-organised representation of the ontology that meets the requirements specification; and *implementation process*, which transforms the design representation of an ontology into an implementation language.

- *Post-development processes*. They are related to the installation, operation, support, maintenance and retirement of an ontology. They are performed after the ontology construction.

2.3) *Integral processes*. They are needed to successfully complete ontology project activities. They ensure the completion and quality of project functions. They are performed at the same time as ontology development-oriented processes and include activities that do not output ontology, but are absolutely necessary to obtain a successful system. They cover the processes of knowledge acquisition, verification and validation, ontology configuration management, documentation development and training.

3) *Methodologies' use*. It refers to the use of the methodology or method in projects, the acceptation of the methodology by other groups to the one that has elaborated the approach, the set of ontologies developed following the approach, the domains of such ontologies, and in which systems they have been used.

4) *Technological support*. It is important to know what tools provide full or partial support to the methodology or method.

References

- M. Fernández-López, A. Gómez-Pérez "Overview and analysis of methodologies for building ontologies". The Knowledge Engineering Review (to be published).
- IEEE 1996. *IEEE Standard for Developing Software Life Cycle Processes*. Std. 1074-1995. IEEE Computer Society. New York (USA).

2.3 Description of methodologies and methods for building ontologies from scratch

This section presents, in chronological order, the most well known approaches for building ontologies both from scratch, or reusing ontologies as they are on the ontology libraries. We will provide a brief description of each approach, presenting who has elaborated it, the proposed steps and activities, the ontologies that have been developed using it, and the applications where such ontologies have been used. We also provide references for each approach.

2.3.1 Cyc method

The Cyc methodology arose from experience of the development of the Cyc knowledge base (KB), which contains a huge amount of common sense knowledge (http://www.cyc.com). The phases to build the Cyc ontology are [Lenat et al., 1990]:

- The first phase proposes to manually code the explicit and implicit knowledge appearing in the knowledge sources without the help of natural language and learning systems. This first phase is carried out by hand, since current natural language systems and learning machines do not handle enough common sense knowledge to search for new common sense knowledge.
- The second phase proposes knowledge codification that is aided by tools using knowledge already stored in the Cyc KB.
- The third phase delegates to the tools the majority of the work. People will only recommend knowledge sources to read, and they will explain the most difficult parts of the text.

Two tasks are carried out in each phase:

1. *Development of a knowledge representation and top level ontology* containing the most abstracts concepts. Terms like *attribute* or *attribute value* are examples of knowledge representation terms, and *thing*, *intangible* or *collection* are other kinds of abstract concepts.
2. *Representation of the rest of the knowledge* using this primitives.

Cyc has been used during the experimentation in the High Performance Knowledge Bases (HPKB), a research program to advance the technology of how computers acquire, represent and manipulate knowledge (http://reliant.teknowledge.com/HPKB/about/about.html).

Until now, this methodology is only used for building the Cyc KB; however, Cyc has different micro-theories showing the knowledge of different domains from different viewpoints. At some areas, several micro-theories can be used, and each micro-theory can be seen from different perspectives and with different assumptions.

Concerning the applications in which Cyc ontologies are used, there are several modules integrated into the Cyc KB and the inference engine. One of these modules is the *system of integration of heterogeneous databases*. This module maps the Cyc vocabulary into the schemas of the databases. That is, the data that are stored on the databases are interpreted according to the Cyc vocabulary. The *Knowledge-Enhanced Searching of Captioned Information* module permits making queries over images using their captions in natural language. Furthermore, the *Guided Integration of Structured Terminology* (GIST) module allows users to import and simultaneously manage and integrate multiple thesauri.

Cyc agents have also been built. All of these agents have a common core with knowledge of Cyc KB, plus particular knowledge in the specific domain of the agent.

Finally, the *WWW Information Retrieval* module, using the natural language tools that also access to the Cyc KB, allows extending the Cyc KB using information available on the Web.

References

- D.B. Lenat, R.V. Guha, Building large knowledge-based systems. (Addison-Wesley Publising Company, Inc. 1990).

2.3.2 Uschold and King's method

This method is based on the experience of developing the *Enterprise Ontology*, an ontology for enterprise modelling processes [Uschold et al.; 95]. This methodology provides guidelines for developing ontologies, which are:

1. *Identify purpose*. It is important to be clear why the ontology is being built and what its intended uses are.
2. *Building the ontology*, which is broken down into three steps:
   2.1. *Ontology capture*, which means:
      - Identification of the key concepts and relationships in the domain of interest, that is, scoping. It is important to centre on the concepts as such, rather than the words representing them.
      - Production of precise unambiguous text definitions for such concepts and relationships.
      - Identification of terms to refer to such concepts and relationships.

      Agreeing on all of the above.

      The authors use a middle-out approach to perform this step and recommend that rather than looking for the most general or the most particular concepts as key concepts, the most important concepts be identified, which will then be used to obtain the remainder of the hierarchy by generalization and specialization.

   2.2. *Coding*. Involves explicitly representing the knowledge acquired in step 2.1 in a formal language.

   2.3. *Integrating existing ontologies*. During either or both of the capture and coding processes, there is the question of how and whether to use ontologies that already exist.
3. *Evaluation*, where the authors adopt the definition of [Gómez-Pérez et al.; 95]: "to make a technical

judgement of the ontologies, their associated software environment, and documentation with respect to a frame of reference ... The frame of reference may be requirements specifications, competency questions, and/or the real world".

4. *Documentation* recommends that guidelines be established for documenting ontologies, possibly differing according to the type and purpose of the ontology.

The most important project developed using this methodology is the Enterprise Ontology, which is a collection of terms and definitions relevant to business enterprises. The ontology was developed under the Enterprise Project by the Artificial Intelligence Applications Institute at the University of Edinburgh with its partners: IBM, Lloyd's Register, Logica UK Limited, and Unilever. (See http://www.aiai.ed.ac.uk/project/enterprise/enterprise/ontology.html)

The most important tool developed using the Enterprise Ontology is the Enterprise Toolset. It uses an agent-based architecture to integrate off-the-shelf tools in a plug-and-play style. The components of the Enterprise Toolset are: a Procedure Builder for capturing process models, an Agent Toolkit for supporting the development of agents, a Task Manager for integration, visualization, and support for process enactment, and an Enterprise Ontology for communication. (See http://www.aiai.ed.ac.uk/project/enterprise for more information).

References

- Uschold, M. King, M. 1995. "Towards a Methodology for Building Ontologies". Workshop on Basic Ontological Issues in Knowledge Sharing.
- Uschold, M.; Grüninger, M. 1996. "Ontologies: Principles Methods and Applications" Knowledge Engineering Review. Vol. 2.
- Gómez-Pérez, A.; Juristo, N.; Pazos, J. *Evaluation and assessment of knowledge sharing technology*. In N. J. Mars (editor), Towards Very Large Knowledge Bases. KBKS 95. IOS Press, Amsterdam, 1995. pp., 289-296.

2.3.3 Grüninger and Fox's methodology

This methodology is based on the experience in developing the TOVE project ontology [Grüninger et al.; 95] within the domain of business processes and activities modelling.

Essentially, it involves building a logical model of the knowledge that is to be specified by means of the ontology. This model is not constructed directly. First, an informal description is made of the specifications to be met by the ontology and then this description is formalized. The steps proposed are as follows (see figure 2.1):

1. *Capture of motivating scenarios*. According to Grüninger and Fox, the development of ontologies is motivated by scenarios that arise in the application. The motivating scenarios are story problems or examples which are not adequately addressed by existing ontologies. A motivating scenario also provides a set of intuitively possible solutions to the scenario problems. These solutions provide an informal intended semantics for the objects and relations that will later be included in the ontology.

   Any proposal for a new ontology or extension to an ontology should describe one or more motivating scenarios, and the set of intended solutions of problems presented in the scenarios.

2. *Formulation of informal competency questions*. These are based on the scenarios obtained in the preceding step and can be considered as expressiveness requirements that are in form of questions. An ontology must be able to represent these questions using its terminology, and be able to characterize the answers to these questions using the axioms and definitions. These are the informal competency questions, since they are not yet expressed in the formal language of the ontology.

   The competency questions are stratified and the response to one question can be used to answer more general questions from the same or another ontology by means of composition and decomposition operations. This is a means of identifying knowledge already represented for reuse and integrating ontologies.

   The questions serve as constraints on what the ontology can be, rather than determining a particular design with its corresponding ontological commitments. There is no single ontology

associated with a set of competency questions. Instead, the competency questions are used to evaluate the ontological commitments that have been made to see whether the ontology meets the requirements.

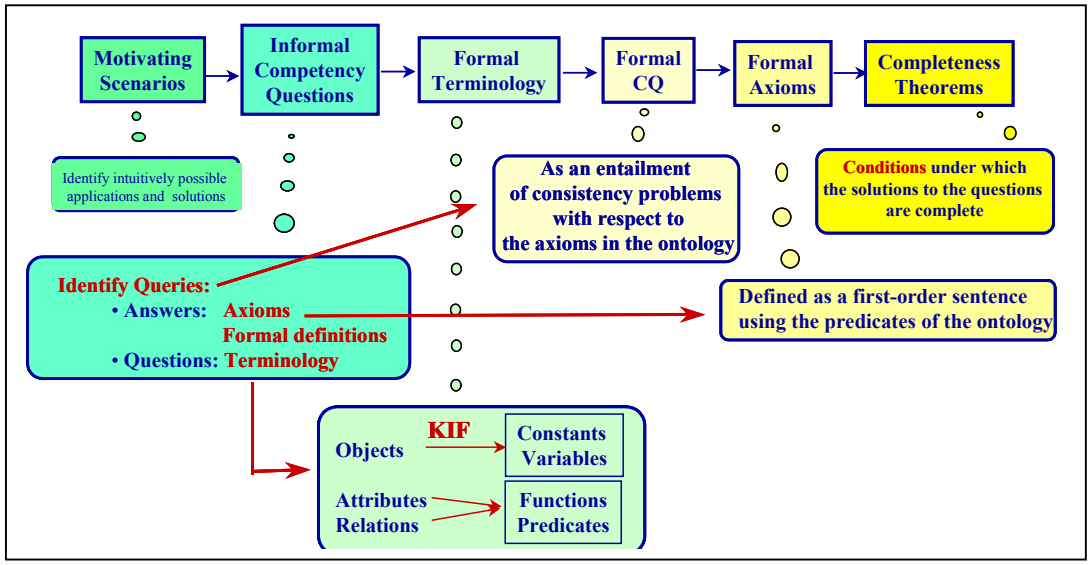

Figure 2.1. Grüninger and Fox's methodology

3. *Specification of the terminology of the ontology within a formal language*. The following steps will be taken:

   3.1. *Getting informal terminology*. Once the informal competency questions are available, the set of terms used can be extracted from the questions. These terms will serve as a basis for specifying the terminology in a formal language.

   3.2. *Specification of formal terminology*. Once informal competency questions have been posed for the proposed new or extended ontology, the terminology of the ontology is specified using a formalism such as KIF [Genesereth et al.; 92]. These terms will allow the definitions and constraints to be later (step 5) expressed by means of axioms.

4. *Formulation of formal competency questions using the terminology of the ontology*. Once the competency questions have been posed informally and the terminology of the ontology has been defined, the competency questions are defined formally.

5. *Specification of axioms and definitions for the terms in the ontology within the formal language*. The axioms in the ontology specify the definitions of terms in the ontology and constraints on their interpretation; they are defined as first-order sentences using axioms to define the terms and constraints for objects in the ontology. Simply proposing a set of objects alone, or proposing a set of ground terms in first-order logic, does not constitute an ontology. Axioms must be provided to define the semantics, or meaning, of these terms.

   If the proposed axioms are insufficient to represent the formal competency questions and characterize the solutions to the questions, then additional objects or axioms must be added to the ontology until it is sufficient. This development of axioms for the ontology with respect to the competency questions is therefore an iterative process.

6. *Establish conditions for characterizing the completeness of the ontology*. Once the competency questions have been formally stated, we must define the conditions under which the solutions to the questions are complete.

This methodology was used to build the TOVE (Toronto Virtual Enterprise) project ontologies at the University of Toronto Enterprise Integration Laboratory. These ontologies constitute an integrated model formalized using first-order logic. The TOVE ontologies include: Enterprise Design Ontology, Project Ontology, Scheduling Ontology, or Service Ontology. For more information, see http://www.ie.utoronto.ca/EIL.

The ontologies built according to this methodology have been used in the applications listed below:

1. *Enterprise Design Workbench*. It is a design environment that allows the user to explore a variety of enterprise designs. The process of exploration is one of design, analysis and re-design, where the workbench provides a comparative analysis of enterprise design alternatives, and guidance to the designer.

2. *Integrated Supply Chain Management Project agents*. The goal is to organize the supply chain as a network of cooperating, intelligent agents, each performing one or more supply chain functions, and each coordinating their actions with other agents. The TOVE virtual enterprise provides the unified testbed used by the agents they built for the major supply chain functions: logistic, transportation, management, etc.

The applications described in this section are still under development. For further information, see http://www.ie.utoronto.ca/EIL.

References

- Genesereth, M. R.; Fikes, R. E. *Knowledge Interchange Format. Version 3.0. Reference Manual*. Computer Science Department. Stanford University. Stanford, California. 1992.
- M. Grüninger, M.S. Fox, Methodology for the design and evaluation of ontologies. Workshop on Basic Ontological Issues in Knowledge Sharing. (Montreal, Canada, 1995).
- M. Uschold, M. Grüninger, Ontologies: Principles Methods and Applications. Knowledge Engineering Review. Vol. 2. (1996). 93-155.

2.3.4 KACTUS method

The work of Bernaras et al. is set within the Esprit KACTUS project [Bernaras et al.; 96]. One of the objectives of the KACTUS project is to investigate the feasibility of knowledge reuse in complex technical systems and the role of ontologies to support it [Schreiber et al.; 95].

This approach to developing ontologies is conditioned by applications development. So, every time an application is built, the ontology that represents the knowledge required for the application is built. This ontology can be developed by reusing others and can also be integrated into the ontologies of later applications. Therefore, every time an application is developed, the following steps are taken:

1. *Specification of the application*, which provides an application context and a view of the components that the application tries to model.

2. *Preliminary design based on relevant top-level ontological categories*, where the list of terms and tasks developed during the previous phase is used as input for obtaining several views of the global model in accordance with the top-level ontological categories determined.

   This design process involves searching ontologies developed for other applications, which are refined and extended for use in the new application.

3. *Ontology refinement and structuring* in order to arrive at a definitive design. The principles of minimum coupling can be used to assure that the modules are not very dependent on each other and are as coherent as possible, looking to get maximum homogeneity within each module.

As experience based on this approach, the authors present the development of three ontologies as a result of the development of the same number of applications. The purpose of the first application is to diagnose faults in an electrical network, the second concerns scheduling service resumption after a fault in the electrical network and the third controls the electrical network on the basis of the above two applications.

References

- A. Bernaras, I. Laresgoiti, J. Corera. Building and Reusing Ontologies for Electrical Network Applications. Proceedings of the European Conference on Artificial Intelligence (ECAI?96). ECAI 96. (1996). 298-302.

- Schreiber, Wielinga and Jasnweijer "The KACTUS view on the 'O' word". *In Proceedings of the Nationala Dutch AI Conference*. NAIC'95.

2.3.5 METHONTOLOGY

This methodology was developed within the Laboratory of Artificial Intelligence at Universidad Politécnica de Madrid. The METHONTOLOGY framework [Fernández et al., 97] [Fernández-López et al., 99] enables the construction of ontologies at the knowledge level. METHONTOLOGY has its roots on the main activities identified by the software development process [IEEE, 96] and knowledge engineering methodologies [Gómez-Pérez et al., 97] [Waterman, 86]. It includes: the identification of the ontology development process, a life cycle based on evolving prototypes, and particular techniques to carry out each activity. As an extension to METHONTOLOGY, a reenginering method was proposed [Gómez-Pérez et al., 99] [Fernández-López et al., 00] (see section 3).
WebODE [Corcho, 02] and ODE [Blázquez et al., 98] are tools built to provide support to METHONTOLOGY. However, other tools can be used following this methodology, for example, Protégé [Protégé, 00] or OntoEdit [Sure, 02].

As we have stated, the ontology life cycle considers the order in which the activities of the ontology development process should be performed. METHONTOLOGY proposes a life cycle based on evolving prototypes for developing ontologies because it allows adding, changing and removing terms in each new version (prototype) of the ontology. For each prototype, METHONTOLOGY proposes beginning with the *specification* of the ontology. Simultaneously with this phase, the knowledge acquisition activity starts. Once the first prototype has been specified, the construction of the conceptual model is built at the *conceptualization* phase. It is like assembling a jigsaw puzzle from the pieces supplied by the knowledge acquisition activity, which is completed during the conceptualization stage. Then, the conceptualization, *formalization* and *implementation* of the knowledge are carried out. After the conceptualization, if some lack in your ontology is detected , you can return to the specification activity to make some modification. If tools like ODE or WebODE are used, the conceptualization model is automatically generated into ontological specification languages through forward translators. Therefore, formalization is not a mandatory activity in METHONTOLOGY.

The life cycle of METHONTOLOGY is shown in figure 2.2, where control, quality assurance, knowledge acquisition, integration, evaluation, documentation, and configuration management are carried out simultaneously with the development activities. However, the stage where the effort for doing knowledge acquisition, integration and evaluation is greater in the conceptualization one, and decreases in the implementation phase. The conceptualization must be very well evaluated to avoid propagating errors in further stages of the ontology life cycle.

METHONTOLOGY also considers that the different activities to be performed during the development of a specific ontology may involve performing other types of activities in other ontologies already built or under construction [Fernández-López et al., 00]. As a consequence, METHONTOLOGY not only consider *intradependencies*, but also *interdependencies*. By intra-dependences we refer to the relationships between activities carried out inside the same ontology, for example, the relationship between knowledge acquisition and conceptualisation in Monatomic Ions. That is, intra-dependences define the ontology life cycle. By inter-dependences we refer to the relationships between activities carried out in different ontologies. Instead of talking about the life cycle of a concrete ontology, we should talk about crossed life cycles of ontologies. The reason is that, most of the times, before integrating an ontology in a new one, the ontology to be reused is modified or merged with other ontologies of the same domain. Let's see this with the example shown in figure 2.3. The development of Chemical Elements in 1996 provoked some changes in Standard Units. For instance, new definitions were included. Such changes were not essential, hence, they could be considered inside the maintenance of the ontology. Several years later, in 1998, Monatomic Ions was built. The development of such ontology caused the merge of the different versions of Chemical Elements, and the reengineering of Standard Units. Therefore, the different versions of Chemical Elements formed only one, whereas a new version of Standard Units appeared as a result of the restructuring of the reengineering process. The old version of Standard Unit was needed because it had been used by other ontologies and applications. As figure 2.3 illustrates, the specification of Monatomic Ions provoked the evaluation of the different versions of Chemical Elements, and the evaluation of Standard Units. Then, during the conceptualization of Monatomic Ions, the integration in this ontology of Chemical Elements led not only to the merging of its different versions but also to the evaluation of such versions and the evaluation of the resulting ontology. The resulting ontology also needed configuration management during all its maintenance. During the

conceptualization of Monatomic Ions, the integration in this ontology of Standard Units led not only to the reengineering of its old version, but also to the configuration management during all its maintenance.

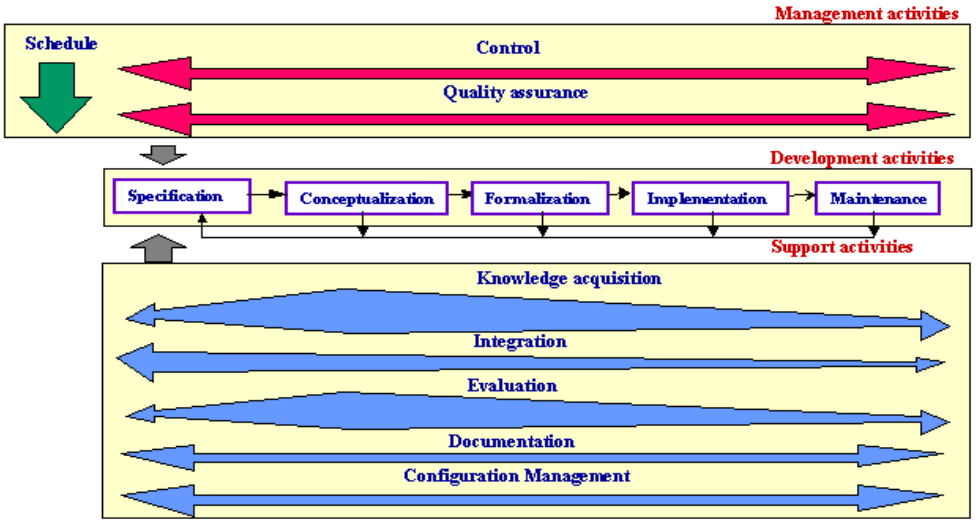

Figure 2.2. Development process and life cycle of METHONTOLOGY

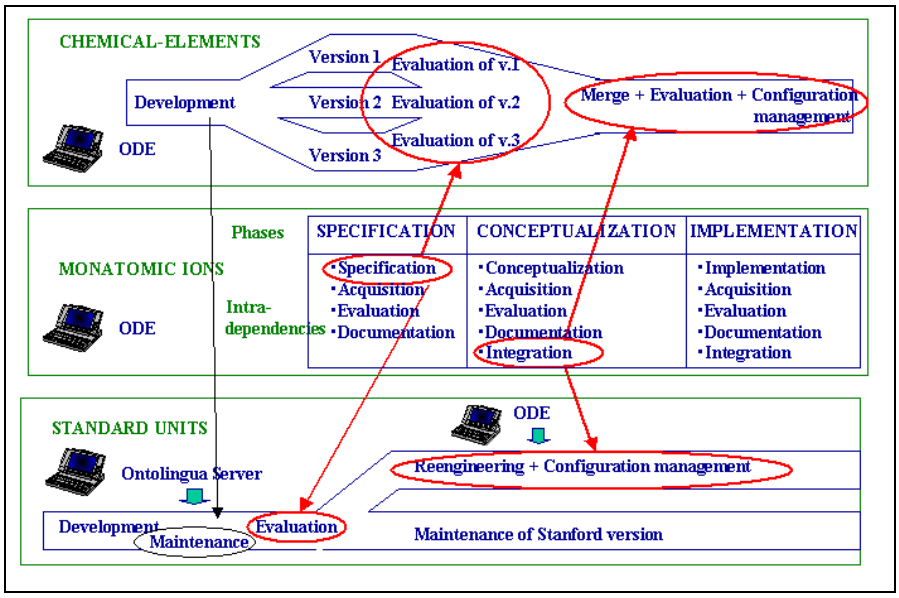

Figure 2.3. Interdependencies and intradependencies in ontology development

These confluences and forking of life cycles call for a global management of ontologies. We claim that, the configuration management of each ontology must not be carried out separately from those in which are integrated. Configuration management must be global and affect simultaneously all the ontologies handled by the group.

METHONTOLOGY has been proposed for ontology construction by the Foundation for Intelligent Physical Agents (FIPA), which promotes inter-operability across agent-based applications[1].

This methodology has been used at UPM for building the following ontologies:

- CHEMICALS [Gómez-Pérez et al., 96] [Fernández-López et al., 99]. It contains knowledge within the domain of chemical elements and crystalline structures.
- Monatomic ions ontology. It gathers information about monatomic ions.

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

- Environmental pollutants ontologies. They represent the methods of detecting the different pollutant components of various media: water, air, soil, etc., and the maximum concentrations permitted of these components, taking into account the legislation in force.
- The Reference-Ontology [Arpírez et al., 98]. It is an ontology in the domain of ontologies that is a kind of ontologies´ yellow pages.
- The restructured version of the $(KA)^2$ ontology [Blázquez et al., 98]. It contains knowledge about the scientific community in the field of Knowledge Acquisition, particularly scientists, research topics, projects, universities, etc.
- Silicate ontology. It models properties in the domain of the minerals and silicates in particular.
- Knowledge management ontologies (KM-LIA). They provide the necessary vocabulary in the domain of people, lessons learned, machines, devices and programs inside the Laboratory of Artificial Intelligence.
- Ontologies developed at IST-1999-10589 MKBEEM project [Léger et al., 00] on travels, cloth catalogues, and lodgings, which have been used as a multilingual platform for e-commerce.
- OntoRoadMap ontology on ontologies, ontology development methodologies, ontology development tools, ontology events (conferences, workshops, etc.), etc.

The applications that use some of the commented ontologies are:

- Onto)2Agent (Arpírez et al., 2000), which is an ontology-based WWW broker about ontologies that uses the Reference-Ontology as a knowledge source and retrieves descriptions of ontologies that satisfy a given set of constraints.
- The OntoRoadMap application2, developed as an evolution of (Onto)2Agent, is an Ontology-based web application that allows the community to register, browse and search ontologies, methodologies, tools and languages for building ontologies, ontology-based applications in areas like the: semantic web, e-commerce, KM, NLP, III, etc., as well as main conferences, workshops and events on that area. It is being used in the IST-2000-29243 Ontoweb thematic network
- Ontogeneration [Aguado et al., 98], a system that uses a domain ontology (CHEMICALS) and a linguistic ontology (GUM) [Bateman et al., 95] to generate Spanish text descriptions in response to queries the domain of chemistry.

References

- Aguado, G.; Bañón, A.; Bateman, J.; Bernardos, S.; Fernández, M.; Gómez-Pérez, A. Nieto, E.; Olalla, A.; Plaza, R.; Sáchez, A. "ONTOGENERATION: Reusing domain and linguistic ontologies for Spanish text generation". Workshop on Applications of Ontologies and Problem-Solving Methods. European Conference on Artificial Intelligence (ECAI'98). Brighton (United Kingdom). 1998.
- Arpírez, J. C.; Gómez-Pérez, A.; Lozano, A. Pinto, H. S. "Reference Ontology and (ONTO)2Agent: The ontology yellow pages". Workshop on Applications of Ontologies and Problem-Solving Methods. European Conference on Artificial Intelligence (ECAI'98). Brighton (United Kingdom). 1998.
- Bateman, J.; Magnini, B.; Fabris, B. "The Generalized Upper Model Knowledge Base: Organization and Use". In N. Mars (Editor). Towards Very Large Knowledge Bases. IOS Press. 1995. pp. 60-72.
- Fernández-López, M.; Gómez Pérez, A.; Rojas Amaya, M. D. 2000. Ontologies crossed life cycles. Workshop on Knowledge Acquisition, Modelling and Management (EKAW). Editor Springer Verlag. Jean Les Pins (Francia). Pp. 65-79.
- Blázquez, M.; Fernández-López, M.; García-Pinar, J.M.; Gómez-Pérez, A. "Building Ontologies at the Knowledge Level using the Ontology Design Environment". Knowledge Acquisition of Knowledge-Based Systems Workshop (KAW). 1998.
- Corcho, O., Fernández-López, M., Gómez-Pérez, A., Vicente, O. WebODE: an integrated workbench for ontology representation, reasoning and exchange. 13th International Conference on Knowledge Engineering and Knowledge Management EKAW02. 2002.
- Fernández-López, M.; Gómez-Pérez, A.; Pazos-Sierra, A.; Pazos-Sierra, J. 1999. Building a Chemical Ontology Using METHONTOLOGY and the Ontology Design Environment. IEEE Intelligent Systems & their applications. January/February PP. 37-46.

[2] http://babage.dia.fi.upm.es/ontoweb/wp1/OntoRoadMap/index.html

- Fernández, M.; Gómez-Pérez, A.; Juristo, N. 1997. METHONTOLOGY: From Ontological Art Towards Ontological Engineering. Symposium on Ontological Engineering of AAAI. Stanford (California).
- Gómez Pérez, A. "Knowledge Sharing and Reuse". In J. Liebowitz (Editor) *Handbook of Expert Systems*. CRC. 1998.
- Gómez-Pérez A, Juristo N, Montes C, Pazos J. *Ingeniería del Conocimiento: Diseño y Construcción de Sistemas Expertos*. Ceura, Madrid, Spain, 1997.
- IEEE 1996. *IEEE Standard for Developing Software Life Cycle Processes*. Std. 1074-1995. IEEE Computer Society. New York (USA).
- Leger, A. and others. 2000. "Ontology domain modeling support for multi-lingual services in E-Commerce: MKBEEM". *ECAI'00 Workshop on Applications of Ontologies and PSMs*. Berlin. Germany.
- *Using Protégé-2000 to Edit RDF*. Technical Report. Knowledge Modelling Group. Technical report. Stanford University. February, 2000. http://www.smi.Stanford.edu/projects/protege/protege-rdf/protege-rdf.html
- Sure Y, Erdmann M, Angele J, Staab S, Studer R, Wenke D (2002) "OntoEdit: Collaborative Ontology Engineering for the Semantic Web". In: Horrocks I, Hendler J (eds) First International Semantic Web Conference (ISWC'02). Sardinia, Italy. Springer Verlag Lecture Notes in Computer Science (LNCS) 2342. Berlin, Germany, pp 221–235
- Waterman DA. *A Guide to Expert Systems*. Addison-Wesley. Masachusets (USA). 1986.

2.3.6 SENSUS method

SENSUS ontology

This is an ontology for use in natural language processing and was developed at the ISI (Information Sciences Institute) natural language group to provide a broad-based conceptual structure for developing machine translators [Knight et al.; 94] [Knight et al.; 95]. Its current content was obtained by extracting and merging information from various electronic sources of knowledge. This process began by merging the PENMAN Upper Model [Bateman et al.; 89] and ONTOS [Nirenburg et al.; 92] (two, very high level linguistically-based ontologies) and the semantic categories from a dictionary by hand to produce an ontology base. WordNet was then merged (again by hand) with the ontology base. A merging tool was then used to merge WordNet with an English dictionary. After, to support machine translation, the result of this merge was then augmented by Spanish and Japanese lexical entries from the Collins Spanish/English dictionary and the Kenkyusha Japanese/English dictionary [Swartout et al.; 97].

SENSUS has more than 70,000 concepts organized in a hierarchy, according to their level of abstraction. It includes terms with both a high and a medium level of abstraction, but, generally speaking, does not cover terms from specific domains. The domain terms are linked with SENSUS in order to build ontologies for particular domains, and any irrelevant terms are pruned in SENSUS.

The method according to the SENSUS approach

When an ontology is to be built in a particular domain, the following steps are taken [Swartout et al.; 97] (see figure 2.4):

1. A series of terms are taken as *seed*.
2. These seed terms are linked by hand to SENSUS.
3. All the concepts in the path from the seed terms to the root of SENSUS are included.
4. Terms that could be relevant within the domain and have not yet appeared are added.
5. Finally, for those nodes that have a large number of paths through them, the entire subtree under the node is sometimes added, based on the idea that if many of the nodes in a subtree have been found to be relevant, then the other nodes in the subtree are likely to be relevant as well. This step is done manually, since it seems to require some understanding of the domain to make the decision. Obviously, very high level nodes in the ontology will always have many paths through them, but it is hardly ever appropriate to include the entire subtrees under these nodes.

An ontology for military air campaign planning has been built using SENSUS. It contains an overview of

the basic elements that characterize air campaign plans, such as campaign, scenario, participants, commanders, etc. [Valente et al.; 99]. This ontology includes ontologies on weapons, systems in general, fuel, etc.

On the basis of SENSUS, knowledge-based applications for the air campaign planning domain have been developed at ISI in conjunction first with the ARPA Rome Planning Institute program and later with the DARPA Joint Forces Air Component Commander program. These include the Strategy Development Assistant [Valente et al.; 99], a tool that provides support for intelligent and guided plan development.

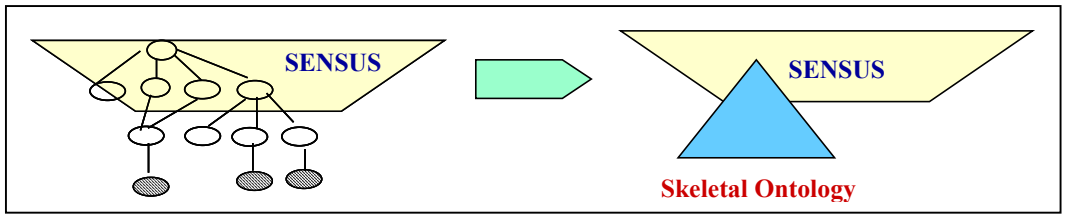

Figure 2.4. SENSUS approach

The result of applying this sequence of steps is a skeletal ontology for a given domain. The method proposes to add by hand new terms to the skeleton.

References

- Bateman, J. A.; Kasper, R. T.; Moore, J. D.; Whitney, R. A. "A General Organization of Knowledge for Natural Language Processing: The Penman Upper Model". USC/Information Sciences Institute. Marina del Rey. 1989.
- Knight, K.; Chancer, I.; Haines, M.; Hatzivassiloglou. V.; Hovy, E. H.; Iida M.; Luk, S.K.; Whitney, R.A.; Yamada, K. 1995. "Filling Knowledge Gaps in a Broad-Coverage MT System". *Proceedings of the 14th IJCAI Conference*. Montreal (Canada).
- Knight, K.; Luck S. 1994. "Building an Large Knowledge Base for Machine Translation". *Proceedings of the American Association of Artificial Intelligence Conference (AAAI-94)*. Seattle (USA).
- Nirenburg, S. and C. Defrise. 1992. "Application-Oriented Computational Semantics*". In: R. Johnson and M. Rosner (eds.) Computational Linguistics and Formal Semantics*. Cambridge, MS: Cambridge University Press. Pages 223-256.
- B. Swartout, P. Ramesh, K. Knight, T. Russ, Toward Distributed Use of Large-Scale Ontologies. Symposium on Ontological Engineering of AAAI. (Stanford, California, March, 1997). 138-148.
- Valente, A.; Russ, T.; McGregor, R.; Swartout, W. "Building and (Re)Using an Ontology of Air Campaign Planning". IEEE Intelligent Systems & their applications. January/February 1999.

2.3.7 On-To-Knowledge Methodology

The On-To-Knowledge methodology was developed and applied in the EU IST-1999-10132 project On-To-Knowledge (see http://www.ontoknowledge.org) for introducing and maintaining ontology based knowledge management applications into enterprises has a focus on Knowledge Processes and Knowledge Meta Processes. While the former process circles around the usage of ontologies, the latter process guides their initial set up.

Figure 2.5: Two orthogonal Processes with feedback Loops

There exist various proposals for methodologies that support the systematic introduction of KM solutions into enterprises. One of the most prominent methodologies is CommonKADS that puts emphasis on an early feasibility study as well as on constructing several models that capture different kinds of knowledge needed for realizing a KM solution. Typically, these methodologies conflate two processes that should be kept separate in order to achieve a clear identification of issues: whereas the first process addresses aspects of introducing a new KM solution into an enterprise as well as maintaining it (the so-called "Knowledge Meta Process"), the second process addresses the handling of the already set-up KM solution (the so-called "Knowledge Process") (see Figure 2.5).

Knowledge Meta Process

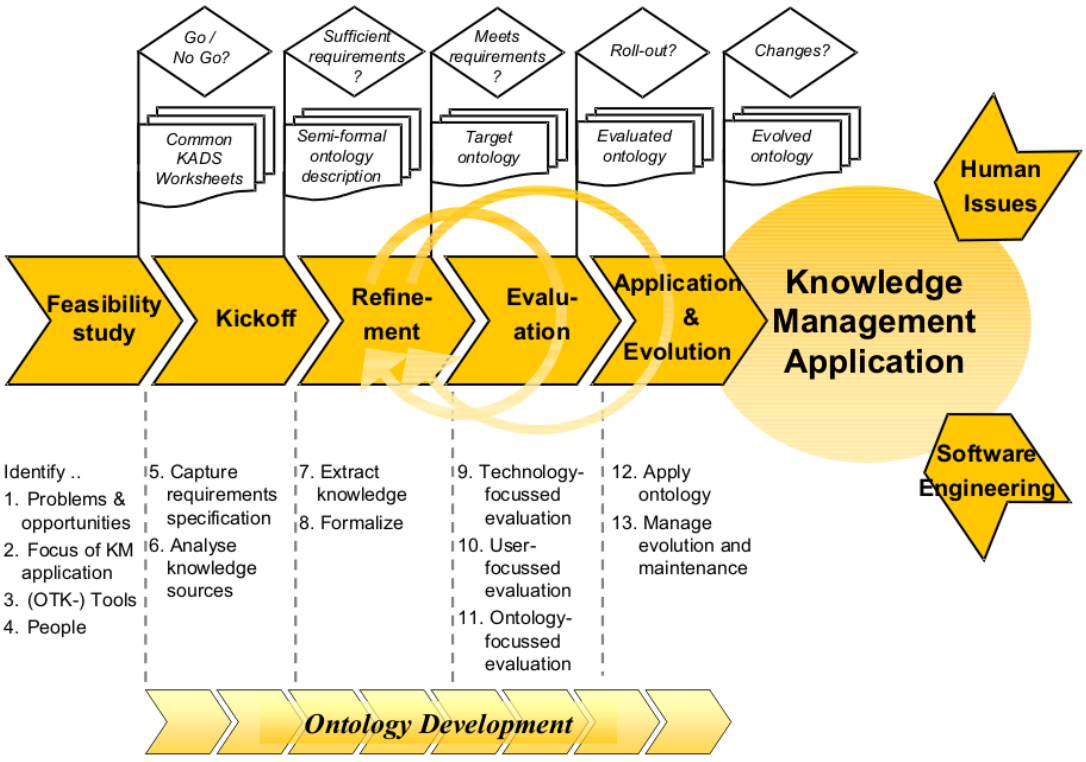

Figure 2.6: The Knowledge Meta Process

The Knowledge Meta Process (cf. figure 2.6) consists of five main steps. Each step has numerous sub-steps, requires a main decision to be taken at the end and results in a specific outcome. The main stream indicates steps (phases) that finally lead to an ontology based KM application. The phases are "Feasibility Study", "Kickoff", "Refinement", "Evaluation" and "Application & Evolution". Below every box depicting a phase the most important sub-steps are listed, e.g. "Refinement" consists of the sub-steps

"Extract knowledge" and "Formalize" etc. Each document-flag above a phase indicates major outcomes of the step, e.g. "Kickoff" results in an "Ontology Requirements Specification Document" etc. Each node above a flag represents the major decisions that have to be taken at the end to proceed to the next phase. The major outcomes typically serve as decision support for the decisions to be taken. The phases "Refinement–Evaluation–Application & Evolution" typically need to be performed in iterative cycles. One might notice that the development of such an application is also driven by other processes, e.g. software engineering and human issues.

Knowledge Process

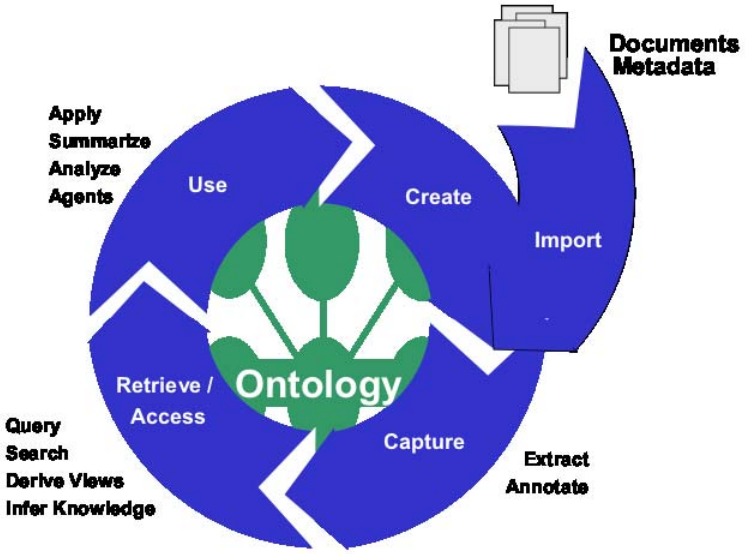

Figure 2.7: The Knowledge Process

Once a KM application is fully implemented in an organization, knowledge processes essentially circle around the following steps (cf. Figure 2.7).

- Knowledge creation and/or import of documents and meta data, i.e. contents need to be created o converted such that they .t the conventions of the company, e.g. to the knowledge management infrastructure of the organization;
- then knowledge items have to be captured in order to elucidate importance or interlinkage, e.g. the linkage to conventionalized vocabulary of the company by the creation of relational metadata;
- retrieval of and access to knowledge satisfies the "simple" requests for knowledge by the knowledge worker; typically, however, the knowledge worker will not only recall knowledge items, but she will process it for further use in her context.

Conclusion

Lessons learned during setting up and employing the methodology in the On-To-Knowledge case studies include:

(i) different processes drive KM projects, but ``Human Issues'' might dominate other ones (as already outlined by Davenport),

(ii) guidelines for domain experts in industrial contexts have to be pragmatic,

(iii) collaborative ontology engineering requires physical presence *and* advanced tool support and

(iv) brainstorming is very helpful for early stages of ontology engineering, especially for domain experts not familiar with modelling.

The described methodology was developed and applied in the On-To-Knowledge project. One of the core contributions of the methodology that could not be shown here is the linkage of available tool support from On-To-Knowledge with the case studies by showing when and how to use tools during the process of developing and running ontology based applications in the case studies.

Also not shown here is the exhaustive tool support by OntoEdit, an advanced ontology engineering environment, that supports each phase of the methodology. We refer to the attached references for further informations.

References

- Y. Sure and R. Studer. "On-To-Knowledge Methodology - Final Version". Institute AIFB, University of Karlsruhe, On-To-Knowledge Deliverable 18, 2002. Available at http://www.aifb.uni-karlsruhe.de/WBS/ysu/publications/OTK-D18_v1-0.pdf
- D. Fensel et al. "On-To-Knowledge: Semantic Web enabled Knowledge Management". In: Special Issue of IEEE Computer on Web Intelligence (WI).
- Y. Sure: "On-To-Knowledge -- Ontology based Knowledge Management Tools and their Application". In: German Journal Künstliche Intelligenz, January 2002 (1/02), Special Issue on Knowledge Management.
- S. Staab, H.-P. Schnurr, R. Studer, and Y. Sure: "Knowledge Processes and Ontologies". In: IEEE Intelligent Systems 16(1), January/Febrary 2001, Special Issue on Knowledge Management.
- Y. Sure, S. Staab, J. Angele. "OntoEdit: Guiding Ontology Development by Methodology and Inferencing". In: Proceedings of the International Conference on Ontologies, Databases and Applications of SEmantics ODBASE 2002, October 28 - November 1, 2002, University of California, Irvine, USA.
- Y. Sure, M. Erdmann, J. Angele, S. Staab, R. Studer and D. Wenke. "OntoEdit: Collaborative Ontology Engineering for the Semantic Web". In: Proceedings of the first International Semantic Web Conference 2002 (ISWC 2002), June 9-12 2002, Sardinia, Italia.
- Th. Lau and Y. Sure. "Introducing Ontology-based Skills Management at a large Insurance Company". In: Modellierung 2002, Modellierung in der Praxis - Modellierung für die Praxis, Tutzing, Deutschland, 25.-27. März 2002.
- Y. Sure and R. Studer: "On-To-Knowledge Methodology". In: *On-To-Knowledge: Semantic Web enabled Knowledge Management.* J. Davies, D. Fensel, F. van Harmelen (eds.), Wiley, to appear 2002
- Y. Sure, M. Erdmann and R. Studer: "OntoEdit: Collaborative Engineering of Ontologies". In: *On-To-Knowledge: Semantic Web enabled Knowledge Management.* J. Davies, D. Fensel, F. van Harmelen (eds.), Wiley, to appear 2002.
- Y. Sure et al. "On-To-Knowledge: Semantic Web enabled Knowledge Management". In: *Web Intelligence.* Ning Zhong (ed.), Springer, to appear 2002.
- S. Staab, R. Studer and Y. Sure: "Knowledge Processes and Meta Processes in Ontology-based Knowledge Management". In: *Handbook on Knowledge Management.* C. W. Holsapple (ed.), Springer, to appear 2002.
- Y. Sure: "A Tool-supported Methodology for Ontology-based Knowledge Management". In: *The Ontology and Modeling of Real Estate Transactions in European Juristictions.* E. Stubkjaer (ed.), International Land Management Series, Ashgate, to appear 2002.

Comparison of approaches against the evaluation framework

Tables 2.1, 2.2, 2.3 and 2.4 contain the comparison information of the different methodologies according to the evaluation framework presented in section 2.2. Table 2.1 presents the approaches classified by groups according the proposed construction strategy, the strategy with respect the application, the use of core ontologies, etc. In table 2.2 we compare the compliance of each approach with respect to the IEEE standard. The rows in this table represent the different approaches, and the columns represent the different processes proposed by the IEEE standard. Each cell of the table can be filled in with three types of values.

The value "described in detail" means that the approach establishes for the considered activity: how to do each task, when to do it, who has to do it, etc. The value "proposed" means that the methodology of the corresponding row identifies the process that is written in the column as a process to be performed during the ontology development process. The value "not proposed" means that public documentation does not mention the non-considered aspect. If the approach partially deals with the process, we include the considered activities. According to this table, there is no methodology with a perfect compliance with respect IEEE standard. Table 2.3 summarises the use of each approach both in ontologies and applications that use such ontologies. And table 2.4 shows the tools used to apply the approach.

Table 2.1: Approaches' proposed construction strategy

Feature	Cyc	Usdhold & King	Grüninger & Fox	KACTUS	METHONTOLOGY	SENSUS	On-To-Knowledge
Life cycle proposal	Evolving prototypes	Non-proposed	Evolving prototypes or incremental?	Evolving prototypes	Evolving prototypes	Non-proposed	Incremental and cyclic with evolving prototypes
Strategy with respect the application	Application independent	Application independent	Application semi-dependent	Application dependent	Application independent	Application semi-dependent	Application dependent
Strategy to identify concepts	Not specified	Middle-out	Middle-out	Top-down	Middle-out	Not specified	Top-down, bottom-up, middle-out – depends on the application
Use of a core ontology	Yes	No	No	No	No	Yes	Depends on th available resources for a Project

Table 2.2: Approaches' proposed ontology development process

Feature			Cyc	Usdhold & King	Grüninger & Fox	KACTUS	METHONTOLOGY	SENSUS	On-To-Knowledge
Project management processes	**project initiation**		Not proposed	Not proposed	Not proposed	Not proposed	Not proposed	Not proposed	Described[3]
	project monitoring and control		Not proposed	Not proposed	Not proposed	Not proposed	***Proposed***	Not proposed	Described
	ontology quality management		Not proposed	Not proposed	Not proposed	Not proposed	Not proposed	Not proposed	Described
Ontology development-oriented processes	**Pre-development processes**	**study of the environment**	Not proposed	Not proposed	Not proposed	Not proposed	Not proposed	Not proposed	Proposed
		feasibility studies	Not proposed	Not proposed	Not proposed	Not proposed	Not proposed	Not proposed	Described
	Development processes	**requirements process**	Not proposed	***Proposed***	***Described in detail***	***Proposed***	***Described in detail***	***Proposed***	Described in Detail
		design process	Not proposed	Not proposed	***Described***	***Described***	***Described in detail***	Not proposed	Described
		implementation process	***Proposed***	***Proposed***	***Described***	***Proposed***	***Described in detail***	***Described***	Described
	Post-development processes	**installation**	Not proposed	Not proposed	Not proposed	Not proposed	Not proposed	Not proposed	Proposed
		operation	Not proposed	Not proposed	Not proposed	Not proposed	Not proposed	Not proposed	Described
		support	Not proposed	Not proposed	Not proposed	Not proposed	Not proposed	Not proposed	Described
		maintenance	Not proposed	Not proposed	Not proposed	Not proposed	***Proposed***	Not proposed	Proposed
		retirement	Not proposed	Not proposed	Not proposed	Not proposed	Not proposed	Not proposed	Not proposed
Integral processes	**knowledge acquisition**		Proposed	***Proposed***	***Proposed***	Not proposed	***Described in detail***	Not proposed	Described
	verification & validation		Not proposed	***Proposed***	***Proposed***	Not proposed	***Described in detail***	Not proposed	Proposed
	ontology configuration management		Not proposed	Not proposed	Not proposed	Not proposed	***Described in detail***	Not proposed	Proposed
	documentation		***Proposed***	***Proposed***	***Proposed***	Not proposed	***Described in detail***	Not proposed	Described
	Training		Not proposed	Not proposed	Not proposed	Not proposed	Not proposed	Not proposed	Described

[3] Each On-To-Knowledge description consists of (a) an explanation of the general ideas of how to carry out the activity; (b) references to authors that have written about such an activity; and (c) the activity software support.

Table 2.3: Approaches' use

Feature	Cyc	Usdhold & King	Grüninger & Fox	KACTUS	METHONTOLOGY	SENSUS	On-To-Knowledge
Projects where the methodology has been used	High Performance Knowledge Bases (HPKB)	Enterprise Project	TOVE	KACTUS	MKBEEM, OntoWeb, Esperonto, founded Spanish projects: "Usos de Ontologías en la Gestión de Conocimientos", "Prototipos de Ontologías para el Medio ambiente", GUME.	Military air campaign planning project in DARPA	On-To-Knowledge, OntoWeb, SemiPort, AIFB Website, COST action "Modeling Real-Property Transactions"
Acceptation by external organisations	Not known	Universidad Politécnica de Cataluña	Not known	Not known	It is the recommended methodology to ontology development by FIPA (Foundation for Intelligence Physical Agents). See chapter titled Ontology Development Process of the FIPA Ontology Service Specification document in http://www.fipa.org/specs/fipa00086/ (last access, 8th November 2002)	Not known	VU Amsterdam (NL), Aidministrator (NL), CognIT as (NO), SwissLife (SL), EnerSearch (SW) BT (UK), Ontoprise GmbH (DE), DFKI Kaiserslautern (DE), Fraunhofer Institute for Integrated Publication and Information Systems (DE), FIZ Karlsruhe (DE)
Ontologies created by the methodology	Cyc	Enterprise Ontology	TOVE	Electrical network ontologies	- CHEMICALS - Monatomic Ions Ontology - Environmental Pollutants - Reference Ontology - Restructured KA2 ontologies - Silicate Ontology - Knowledge management ontologies (KM-LIA) - MKBEEM ontologies - OntoRoadMap ontologies - Esperonto ontologies	Military air campaign planning ontologies	Skills Management @ SwissLife, Virtual Organization @ EnerSearch, OntoShare @ BT, OntoWeb Portal, AIFB Portal
Domains of the ontologies	Cyc have different micro-theories	Enterprise	Enterprise	Electrical network	- Chemical - Environment - Ontologies - Knowledge management - Computer Science - Travels - Etc.	Military air campaign	See above
Applications that use such ontologies	- System of integration of heterogeneous databases - Knowledge-Enhanced Searching of Captioned Information - Guided Integration of Structured Terminology	Enterprise Toolset	Enterprise Design Workbench Supply Chain Management Project	Diagnosis and and service resumption in electrical networks	- Ontogeneration - OntoRoadMap - (Onto)2Agent - ODEClean - MKBEEM prototype	Not known	See above

Table 2.4: Approaches' technological support

Feature	Cyc	Usdhold & King	Grüninger & Fox	KACTUS	METHONTOLOGY	SENSUS	On-To-Knowledge
Tools that provide support	Cyc tools	No specific tool	No specific tool	No specific tool	- ODE - WebODE (integrated ontology development platform)	No specific tool (usually OntoSaurus)	OntoEdit with plugins: OntoMap, OntoFiller, Domain Lexicon, Inferencing, etc. OntoMat-Annotizer, OntoBroker, MindManager2002 Business Edition, Sesame, Spectacle, OntoShare, Ontology Middleware Module (OMM), …

3 Method for reengineering ontologies

Ontological reengineering [Gómez-Pérez et al., 1999] is the process of retrieving and mapping a conceptual model of an implemented ontology to another, more suitable conceptual model, which is re-implemented. Until now, the only known proposal for ontology reengineering is that elaborated by the Ontology Group of Artificial Intelligence Lab. at UPM.

3.1 Method for reengineering ontologies of the Ontology Group at UPM

A proposal of a method for reengineering ontologies is presented in figure 3.1 and adapts Chikofsky's software reengineering schema [Chikofsky, 1990] to the ontology domain. Three main activities were identified: reverse engineering, restructuring and forward engineering. Figure 3.2 pictures in detail an organisational chart showing the activities performed during the reengineering process and the documents generated in each step. A description of each step is presented:

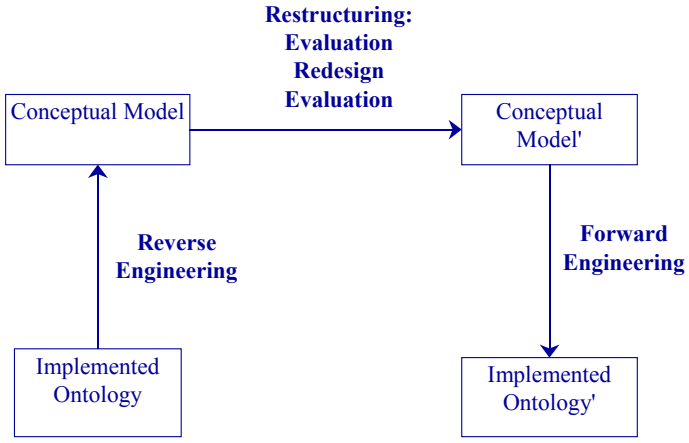

Figure 3.1. Ontology reengineering process

Reverse Engineering: its objective is to output a possible conceptual model on the basis of the code in which the ontology is implemented. For the purpose of building a conceptual model, the set of intermediate representations proposed by METHONTOLOGY are used. In fact, this reengineering methodology can be considered an extension of the METHONTOLOGY framework.

Restructuring: the goal of restructuring is to reorganised the initial conceptual model into a new conceptual model which is built bearing in mind the use of the restructured ontology by the ontology/application that reuses it, which means that there is no way of assuring that the restructured ontology will be a hundred per cent valid for all the ontologies that reuse the restructured knowledge. The restructuring activity contains two phases: analysis and synthesis. The analysis phase includes evaluation (steps 2 to 5 of figure 3.2), whose general aim is to evaluate the ontology [Gómez-Pérez et al., 1995], that is, to check that the hierarchy of the ontology and its classes, instances, relations and functions are complete, consistent (there are no contradictions), concise (there are no explicit and implicit redundancies) and syntactically correct. The synthesis phase (step 6 of figure 3.2) seeks to correct the ontology after the analysis phase and document any changes made. Hence, activities that are related with *configuration management* arise in that context, whose goal is to keep a record of ontology evolution and strict change control.

A series of documents will be generated, which can be divided into three groups: (1) analysis documents, including a list of anomalies (problems, errors, omissions, ambiguities, etc.) encountered and detected in steps 2 and 4; (2) configuration management documents, which include reports related to the changes made in steps 3 and 5 on the basis of the set of errors identified in the analysis documents. These documents include: description, need and effects of the change, possible alternatives, justification of the

selected alternative, date of the change, etc.; and (3) synthesis documents, including the taken actions and observed criteria in step 6.

Forward Engineering: the objective of this step is to output a new implementation of the ontology on the basis of the new conceptual model.

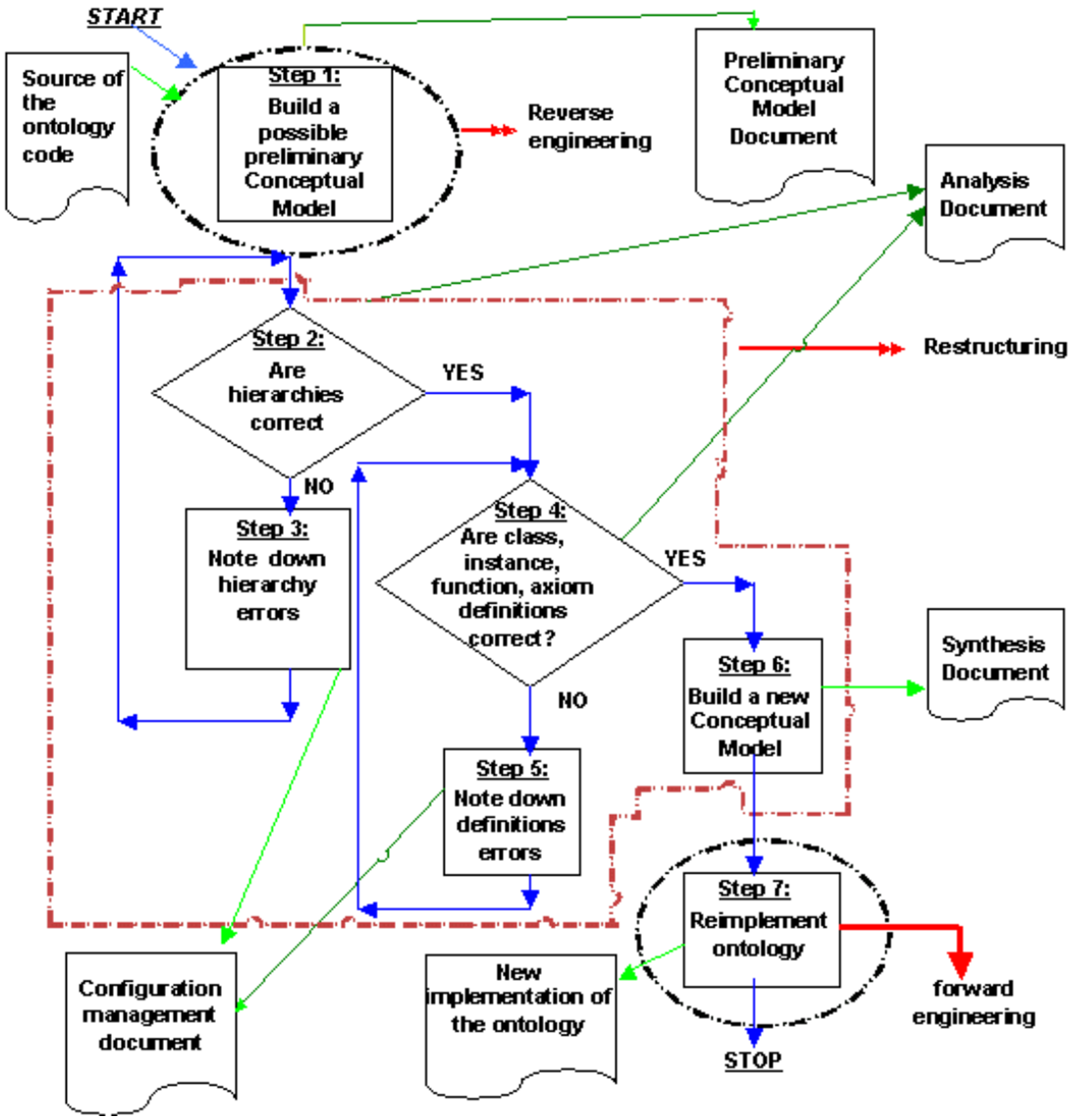

Figure 3.2. Ontological reengineering activities

Finally, to keep control of the changes made, configuration management should be performed not only during the restructuring process, as we said before, but also during the whole process. For the purpose of assuring information about the evolution of the ontology, a rigorous change control must be performed throughout the restructuring phase. The goal is to have all the changes documented, detailing the changes made, their causes and effects. The configuration management documents can rule out incorrect decision making, if they state the courses of action to be taken at any time, and justify the choice of one rather than another. Change control also helps end users to determine which version of the ontology they require for their system or for the new ontology they are to develop.

Change control starts with a petition for change, followed by the classification and registration, approval or initial rejection and evaluation of the change petition, submission of the change report to the Change Control Committee, performance of the change and certification that the change was made correctly. It ends when the result is reported to the person who proposed the change.

Some of the reengineered ontologies are Standard Units (Gómez-Pérez et al. 1999) and ontologies that are produced at the IST-1999-10589 MKBEEM project (Leger et al. 2000). In the first case, measure units of the International System were needed to build the ontology Monatomic Ions. This fact leads to the reusing of Standard Units (Gruber et al., 1994) ontology, which is in the Ontolingua Server (Farquhar et al., 1997).

References

- Chikofsky, E.J.; Cross II, J.H. 1990. "Reverse Engineering and design recovery: A taxonomy". *Software Magazine*. January/February (Volume 7, Number 1). Pages 13-17.
- Gómez-Pérez, A; Rojas, M. D. 1999. "Ontological Reengineering and Reuse". European Knowledge Acquisition Workshop (EKAW).
- Gómez-Pérez, A.; Juristo, N.; Pazos, J. "Evaluation and assessment of knowledge sharing technology". In N. J. Mars (editor), Towards Very Large Knowledge Bases. KBKS 95. IOS Press, Amsterdam, 1995. pp., 289-296.

4 Methods for cooperative construction of ontologies

4.1 Introduction

An ontology is a shared and common understanding of some domain. Righ now, emphasis is put on consensus on the content of ontologies, in the sense that a group of people agrees on the formal specification of the concepts, relations, attributes and axioms that the ontology provides. However, the problems of how to jointly construct an ontology (with a group a people) and how to distributively construct an ontology (with people at different locations) are still unsolved. Euzenat identified the following problems[4] concerning collaborative construction of ontologies: management of the interaction and the communication between people; data access control; recognition of a moral right about the knowledge (attribution); detection and management of errors; and concurrent management and modification of the data. The consequence of these problems is that there are a few detailed proposals about how to collaboratively build ontologies. Nevertheless, since several years ago, the main proposal are: CO4, for collaborative construction of KB's at INRIA; and the approach used in ontologies building at the Knowledge Annotation Intiative of the Knowledge Acquisition Community, also know as (KA)2 initiative.

4.2 Description of methods for cooperative construction of ontologies

We have described in this deliverable Co_4, a protocol for collaborative construction of consensual knowledge bases in the Web, and $(KA)^2$, obtained from the experience in ontology development in the Knowledge Annotation Initiative of the Knowledge Acquisition community.

4.2.1 Co_4: Collaborative construction of consensual knowledge bases

CO4 (for Collaborative construction of consensual knowledge bases) is an infrastructure enabling the collaborative construction of a knowledge base through the web. The consensual knowledge base is meant to represent the consensus among a community about a domain to model (Euzenat, 1995). The knowledge base is accessible from a HTTP client and can be consulted or edited by authorised people.

A key idea in the approach taken here is that formally expressed knowledge serves a variety of purposes including knowledge and data search, but above all knowledge elaboration (i.e. the organisation and formalisation of knowledge). Knowledge elaboration can be though of as a social process, involving the cooperation of a variety of agents. The CO4 system aims at supporting the elaboration with the help of knowledgeable people, i.e. by enforcing a kind of peer review process on the modifications attempted.

Formality and consistency require more strictness in the protocol than pure peer-reviewing because it is not possible to deal with an inconsistent knowledge base contrary to a paper journal in which the articles do not have to be consistent. This justifies the consensus requirement in which a modification, for being accepted, must have been agreed by all members.

The task of the editor-in-chief is automatically carried out according to a formal protocol, which handles the communication between knowledge bases. The protocol has been fully described and proved consistent, consensual, live and fair under reasonable assumptions (Euzenat, 1997). CO4 can be thought of as a formalised scientific journal (both in its content and in its functioning).

In order for the proposal to be feasible, the cooperators do not directly modify the knowledge base but their own personal workspace. In CO4, anyone is viewed by the system as a knowledge base. In order to build a consensual knowledge base, the individual knowledge bases must be linked together. Knowledge bases are organised into a tree whose leaves are *user knowledge bases* and whose intermediate nodes are called *group knowledge bases*. Each group knowledge base represents the knowledge consensual among its sons (called *subscriber knowledge bases*).

As soon as the knowledge base is part of a group knowledge base, it receives its complete contents, it is entitled to give its opinion on all submissions currently under examination and it is allowed to submit knowledge. A group knowledge base sends to its subscribers messages in order to broadcast a change

[4] (see http://www.inrialpes.fr/sherpa/papers/euzenat98c.html)

accepted by everyone and calls for comments in order to establish whether a change must be committed or not. A (group or individual) knowledge base sends to its group knowledge base changes that it wants the group knowledge base to integrate.

When subscribers are sufficiently confident about some pieces of knowledge, they can submit them to their group knowledge base. This proposal is then submitted to the other subscribers as a call for comments. In response, users must answer by one of the following: *accept* when they consider that the knowledge must be integrated in the consensual knowledge base, *reject* when they do not, and *challenge* when they propose another change. When the group knowledge base has gathered enough comments, three cases may happen:

- All of them agree to accept the modification, then the modification is committed into the group knowledge base and broadcast to every subscriber knowledge base;
- One of them rejects the proposal, then the changes are not committed and the comments provided by the rejecter are sent to the submitter;
- One submitter sends a counter-proposal, then the call for comments is replaced by a call for comments about all the available proposals.

The CO4 protocol applies to several levels: the group knowledge bases can be grouped together into a more important group knowledge base and so on.

References

- J. Euzenat, Building consensual knowledge bases: context and architecture, in N. Mars (ed.), Towards very large knowledge bases, IOS press, Amsterdam (NL), pp143-155, 1995
- J. Euzenat, C. Chemla, B. Jacq, A knowledge base for *D. melanogaster* gene interactions involved in pattern formation, Proc. 5th international conference on intelligent systems for molecular biology, Halkidiki (GR), pp108-119, 1997
- J. Euzenat, A protocol for building consensual and consistent repositories, Research report 3260, INRIA Rhône-Alpes, Grenoble (FR), septembre 1997, submitted for publication, 46p.

4.2.2 $(KA)^2$ method

The goal of the Knowledge Annotation Initiative of the Knowledge Acquisition community, also acknowledged as the $(KA)^2$ initiative, is to model the knowledge-acquisition community using ontologies developed in a joint effort by a group of people at different locations using the same templates and language. The $(KA)^2$ ontology will form the basis to annotate WWW documents of the knowledge acquisition community in order to enable intelligent access to these documents. $(KA)^2$ is an open-joint initiative where the participants are actively involved in the distributive ontological engineering development.

The current conceptual model of the $(KA)^2$ ontology consists of seven related ontologies: an organization ontology, a project ontology, a person ontology, a publication ontology, an event ontology, a research-topic ontology and a research-product ontology.

In the collaborative and distributed process that was used to build the ontologies, two kinds of people were involved. They were called: *Ontopic agents* and *Ontology coordinating agents*.

Ontopic agents were research groups that have a deep knowledge about some topics of interest. There exist about 15 groups of ontopic agents, each group focusing on a particular topic of the KA ontology. Their aim was to establish a consensual ontology of the KA community, hence they had to have a shared vision of the topic which is being represented.

The majority of the $(KA)^2$ ontologies (organisations, projects, people, publications, events, and research-products) were directly developed by the ontology coordinating agents. To build these ontologies, they created their conceptual structure and identified their main concepts, taxonomies, relations, functions, attributes and axioms, delegating the addition of instances to the community. However, the development of the Research-Topic ontology was carried out jointly by ontopic agents, who were experts of the research topic domain and the ontology coordinating agent. To ease the process of building the Research-Topic ontology, the ontology coordinating agent distributed a template among the Ontopic agents, which used e-mail in their intra-communication and also to send their results to the

coordinating agents. The ontology was generated from the knowledge introduced via the template. Once the ontology coordinating agents got all the portions of the ontologies from the ontopic agents, they integrated them. In this case, the integration process was not difficult since all the ontopic agents used the same pattern.

A first release of the $(KA)^2$ ontology was built in Flogic (Kifer et al., 1995), which is the ontology language used by Ontobroker (Fensel et al., 1998). A decision was made to translate the whole ontology to Ontolingua to make it accessible to the entire community through the Ontolingua Server. Translation between FLogic and Ontolingua formats of the ontology was performed automatically by translators, which form part of ODE.

References

- Stefan Decker, Michael Erdmann, Dieter Fensel, and Rudi Studer: Ontobroker: Ontology Based Access to Distributed and Semi-Structured Information. In R. Meersman et al. (eds.): Semantic Issues in Multimedia Systems. Proceedings of DS-8. Kluwer Academic Publisher, Boston, 1999, 351-369.

5 Ontology merging methods and methodologies

5.1 Introduction

Ontology merge has become a key point for the last years. On the one hand, ontology merge in design time is very important, since the merger of companies or organisations in general can lead to a merge of their ontologies. Even you can have the necessity of merging several ontologies to have another with better quality.

On the other hand, ontology merging in run-time can be also crucial. In fact, ontologies have become increasingly common on the World ide Web where they provide semantics for annotations in Web pages (Noy et al., 2001). Moreover, the heterogeneity of information in the Web can provokes to deal with different ontologies in similar domains. Such diversity must coexist with the interaction between systems. An option to make compatible the diversity and the generality is to establish mappings between ontologies, and to merge them in run-time (Mena et al.; 2001).

As a consequence of the situation shown in the former paragraphs, several ontology merging methodologies have appeared.

5.2 Description of methods and methodologies for ontology merging

This section presents the most well known approaches for ontology merging. Section 5.2.1 presents ONIONS, focusing on its ontology merging proposal. Section 5.2.2 presents PROMPT, an algorithm used in Protégé 2000 (see OntoWeb deliverable 1.3) for ontology merging. Section 5.2.3 presents FCA-Merge, which merges ontologies using documents on the same domain as the ontologies to be merged. Section 5.2.4 presents IFOM, partially inspired in FCA-Merge. Finally, section 5.2.5 presents MOMIS, which merges ontologies by means of ontology clustering.

5.2.1 ONIONS

ONIONS (*ONtological Integration Of Naïve Sources)* [Gangemi et al., 1999] is a methodology for conceptual analysis and ontological integration or merging of terminologies.

ONIONS aims to provide extensive axiomatization, clear semantics, and ontological depth to the domain terminologies that are to be integrated or merged.

Extensive axiomatization is obtained through a careful conceptual analysis of the terminological sources and their representation in a logical language with a rigorous semantics.

Ontological depth of the analysis is obtained by reusing a library of foundational ontologies, on which the axiomatization depends. Such library may include multiple choices among partially incompatible ontologies. Consequently, the tools that implement ONIONS should support enough expressivity and classification services.

Ontological analysis and merging of terminologies in ONIONS are carried out in two phases: (I) re-engineering of ontology building data, and (II) ontology merging (described here).

The re-engineering phase addresses the *extraction*, *formatting*, *analysis*, and *formalization* of conceptually relevant data from the sources [Gangemi et al., 1999]. Extraction and formatting are performed by wrapping the relevant data, after a preliminary evaluation of the data types. Analysis consists in linguistic and conceptual techniques for expliciting the intended meaning of relevant data (usually terms). Finally, analyzed data are formalized in a logical language.

If the domain ontologies obtained in the re-engineering process are heterogeneous (the usual case), they can be either integrated or merged. *Integration* only allows for some partial interoperability depending on how detailed the core ontology is.

Merging amounts to create a new ontology having the same extensional coverage of the merged ones, but a different amount and/or distribution of properties and axioms (a different intension).

Merging is also needed to build the *core* domain ontology when none is available (see domain analysis). The core ontology is built by reusing existing sources, and by merging them with a foundational ontology. The result is a *reference* ontology for a domain.

If domain ontologies haven't been built or upgraded wrt reference ontology, the amount of work to be carried out for merging them is much harder. As a matter of fact, without reference ontology, we can only use the structural distribution of axioms and naming similarity as merging strategies, since we cannot base the merging work on some shared set of relations and concepts.

Alternatively, we can apply ONIONS domain analysis to the ontologies to be merged, and then merge them with the following guidelines.

Assuming that the domain ontologies have been built according to the re-engineering guidelines described in the phase I of ONIONS, they can be *logically integrated* by creating the union of their vocabularies, and comparing the axiomatization of each one to the others [Calvanese et al., 2001].

Purely logical integration is only the backbone of ontology merging: as a matter of fact, current methodologies [e.g. Palopoli et al., 1999] *assume* the existence of a domain repository for managing synonymy, subsumption, and polysemy. This is a practical problem, since domain repositories have nowadays a poor quality from the vantage point of conceptual modelling. ONIONS merging methodology provides guidelines to judge on synonymy, subsumption, and polysemy.

Within ONIONS, plain logical integration works fine for *defined* concepts, but it may originate a redundant ontology when applied to *primitive* concepts with distinct names. This is quite obvious, since defined concepts are completely described by their axioms, which are comparable against the common core ontology. On the other hand, primitive concepts are only partly described by axioms (if any), then we cannot draw any conclusion on their equivalence, and the integration will keep maintaining a *redundant* ontology. A remedy to redundancy is the axiomatic enrichment of taxonomies.

Moreover, some possible relations between primitive concepts across the integrated ontologies may remain implicit, thus originating also an *incomplete* ontology. A remedy to incompleteness is the analysis of polysemy networks and the production of core axiom schemata.

Axiomatic enrichment and axiom schemata are complementary, since schemata can be instantiated to enrich taxonomies.

ONIONS merging strategy reuses techniques from domain analysis for extracting axioms from bare taxonomies. In particular, the situation of a lightweight ontology or a simple taxonomy, showing sets of concepts subsumed by the same parent concept without any explicit, common criterion, is similar to the situation of an integrated ontology that is not yet merged.

Since ONIONS requires the use of foundational and core ontologies to axiomatize alternative theories, the integration of a concept C from an ontology O_1 is performed by comparing C with the concepts $D_{1,\ldots,n}$ already present in the evolving ontology library **L**, whose ontology set $O_{1,\ldots,n}$ contains at least a significant subset of foundational and core ontologies at that state in the evolution of **L**. Usually it is case that **L** integrates an alternative domain ontology O_2 that is going to be merged with O_1.

In the following, *synonymy* extraction, taxonomy *enrichment* and *polysemy management* techniques are briefly explained.

1. Synonymy extraction. C's name is synonym with a name of a D_i. Synonymy is the converse of homonymy and occurs when two concepts with different names have both the same intended model and the same domain. This is the less controversial activity in ontology merging. Several heuristics to extract candidate synonyms can be used based on naming similarity, lexical cooccurence, property sharing (this can be derived by logical integration), and available lexical repositories. Candidates can then be evaluated through domain analysis techniques. In case of sibling synonym pairs, taxonomy enrichment can be helpful. Synonyms are usually included in the set of lexical realizations of the concept.
2. Taxonomy enrichment. Any two siblings resulting from the logical integration of two ontologies should be differentiated explicitly, unless they can be merged as synonyms. In the general case, and without the pre-processing offered by ONIONS domain analysis, the task would be very hard, since the intended meaning of two heterogeneous siblings can even be distributed into two disjoint concepts within a foundational or core ontology. But in our case, provided that the ontologies have been upgraded during the domain analysis, we can require that for any set of sibling concepts from heterogeneous domain ontologies, the conceptual difference between each of them is inferred, and such difference is formalized by axioms that reuse - if available - the relations and concepts already in the library. If some concept or relation is not available to represent the difference, it is added to the library. In other words, this activity can be described as the creation of an axiom schema A for a set of

concepts $\varphi_{1,\ldots,n}$ under a common parent ψ, where A has either the existential form:

$[\forall x.\ \varphi_i(x) \rightarrow (\psi(x) \wedge (\exists n(y_{1,\ldots,a}).\ (\Re_a(x,y_{1,\ldots,a}) \wedge (\chi_1(y_1) \wedge \ldots \wedge \chi_a(y_a)))))]$, or the universal form:

$[\forall x.\ \varphi_i(x) \rightarrow (\psi(x) \wedge (\forall(y_{1,\ldots,a}).\ (\Re_a(x,y_{1,\ldots,a}) \rightarrow (\chi_1(y_1) \wedge \ldots \wedge \chi_a(y_a)))))]$,

where $\exists_n$ is an existential cardinality constraint on the arguments of a χ_i; $\Re_a$ is a conceptual relation of arity *a*, and χ_i is a concept. Universal quantification is intended on the domain of interpretation targeted by the ontology. The axiom schema is *instantiated* for a concept φ_1 by specializing or instantiating the concept(s) χ_i.

2.1. This activity can use the same techniques as those from ONIONS domain analysis, for example an axiom schema from the core ontology can be inductively inherited. As far as merging is concerned, it is aimed at extracting the *minimal* axiom schemata that differentiate the siblings (e.g. distinguishing goods by color, weight, parts, functionalities, etc.).

2.2. Since the main goal is extracting axioms, if there are no terminological clues, knowledge elicitation has to be carried out from experts or texts.

2.3. Often all the siblings of a common parent can be differentiated by only *one* axiom schema. In other cases, several schemata should be provided for different subsets of siblings: this could suggest the creation of new *intermediate* concepts between the common parent and each subset of siblings that share the same axiom schema (e.g. “goods by color”, “goods by function”, etc.).

3. Polysemy management. For any set of polysemous senses of a concept name used in different ontologies, different concepts are stated and placed within the library according to their domain and to the available modules. Polysemy occurs when two concepts with overlapping or disjoint intended models have the same name (or a pair of accepted synonyms). Several kinds of conceptual polysemy may appear, besides homonymy: The general interesting case is multiple subsumtpion, e.g., concept C is synonymous with concept C_1 in **L**, C is subsumed by concept D in **L**, and C_1 is subsumed by concept D_1 in **L**. This may not happen when synonymy is generated by common properties of defined concepts. This is also the most interesting part for polysemy management; several sub-cases may occur, and each one is handled appropriately:

3.1. Secondary taxonomy: either D or D_1 are role-like concepts that share a common super-concept E. A role-like concept is a 'secondary' concept, whose axioms display transitory or functional features of entities.

3.2. Systematic polysemy 1: D and D_1 are two related, sibling concepts that are linked by a hidden relation that is relevant for the domain. By *related* here we mean two concepts that can appear in an axiom schema [Gangemi et al., 2000].

3.3. Systematic polysemy 2: D and D_1 are two related, non-sibling concepts that are linked by a hidden relevant relation.

In all these cases, an axiom schema is built, or it can be found to be a specialization of an existing axiom schema in the reference ontology. Accordingly, C and C_1 are subsumed by different concepts within the schema.

Discussion

ONIONS started in 1993 as an alternative to the bottom-up ontology construction methodology applied in the GALEN project. Since then, ONIONS has been applied to the construction of a medical core ontology (ON9, [Gangemi et al. 1999]), to an ontological mining of the UMLS repository [ibid.], to the integration of clinical guideline standards [Pisanelli et al. 2000], and more recently to web catalogues, legal regulations [Gangemi et al. 2001-b], banking procedures, a revisitation of WordNet [Gangemi et al. 2001-a], fishery terminologies merging [Gangemi et al. 2002], etc.

While the focus of ONIONS has been maintained, the use of foundational and core ontologies has been precised in the last years, by admitting an extensively axiomatized upper-level, currently DOLCE [Masolo et al., 2002], and axiom schemata for core ontology development [Gangemi et al., 2001-b].

References

- Calvanese D, De Giacomo G, Lenzerini M.: A Framework for Ontology Integration. Proceedings of 2001 Int. Semantic Web Working Symposium (SWWS 2001) (2001)
- Gangemi A, Pisanelli DM, Steve G: An Overview of the ONIONS Project: Applying Ontologies to the Integration of Medical Terminologies. *Data and Knowledge Engineering*, 1999, vol.31, pp. 183-220 (1999).
- Gangemi A, Pisanelli DM, Steve G, Understanding Systematic Conceptual Structures in Polysemous Medical Terms, in: J Marc Overhage (ed.), "Converging Information, Technology, and Health Care", *Proceedings of the 2000 AMIA Annual Symposium* (2000).
- Gangemi A, Guarino N, Oltramari A, "Conceptual Analysis of Lexical Taxonomies: The Case of WordNet Top-Level", in: C Welty, B Smith (eds.), *Proceedings of the 2001 Conference on Formal Ontology and Information Systems,* Amsterdam, IOS Press (2001).
- Gangemi A, Pisanelli DM, Steve G, An Ontological Framework to Represent Norm Dynamics, in: R Winkels (ed.), *Proceedings of the 2001 Jurix Conference, Workshop on Legal Ontologies*, University of Amsterdam (2001).
- Gangemi A, Fisseha F, Pettman I, Pisanelli DM, Taconet M, Keizer J, A Formal Ontological Framework for Semantic Interoperability in the Fishery Domain, in ECAI02 Workshop on Semantic Interoperability, H. Stuckenschmidt (ed.), (2002).
- Guarino, N. & Welty, C. (2002). Evaluating Ontological Decisions with OntoClean. Communications of the ACM, 45(2), (61-65).
- Masolo, C., Gangemi, A., Guarino, N., Oltramari, A., and Schneider, L.: WonderWeb Deliverable D17: The WonderWeb Library of Foundational Ontologies, (2002)
- Palopoli L, Saccà D, Ursino D. A novel approach to cooperative information system construction and management. Intl. Workshop on Intelligent Information Systems, 109-126, Brighton (UK) (1998).
- Pisanelli DM, Gangemi A, Steve G, The Role of Ontologies for an Effective and Unambiguous Dissemination of Clinical Guidelines, in R Dieng, O Corby (eds.), "Knowledge Engineering and Knowledge Management. Methods, Models, and Tools", Berlin, Springer-Verlag, pp. 129-139 (2000).

5.2.2 PROMPT

PROMPT[Noy and Musen, 2000] is an algorithm to ontology merging and alignment that is thought to be semi-automated. Therefore, it is implemented in a plugin for Protégé-2000[5], which performs some tasks automatically and guides the user in performing other tasks for which his intervention is required. The plugin determines possible inconsistencies in the state of the ontology, which result from the user's actions, and suggests ways to remedy these inconsistencies (see figure 5.1). PROMPT is based on an extremely general knowledge model and therefore can be applied across various platforms.

PROMPT 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 to the classes that the user said are 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)

[5] http://protege.stanford.edu

In addition to providing suggestions to the user, PROMPT identifies conflicts. Some of the conflicts that PROMPT identifies 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. AnchorPROMPT [Noy and Musen, 2001] 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.

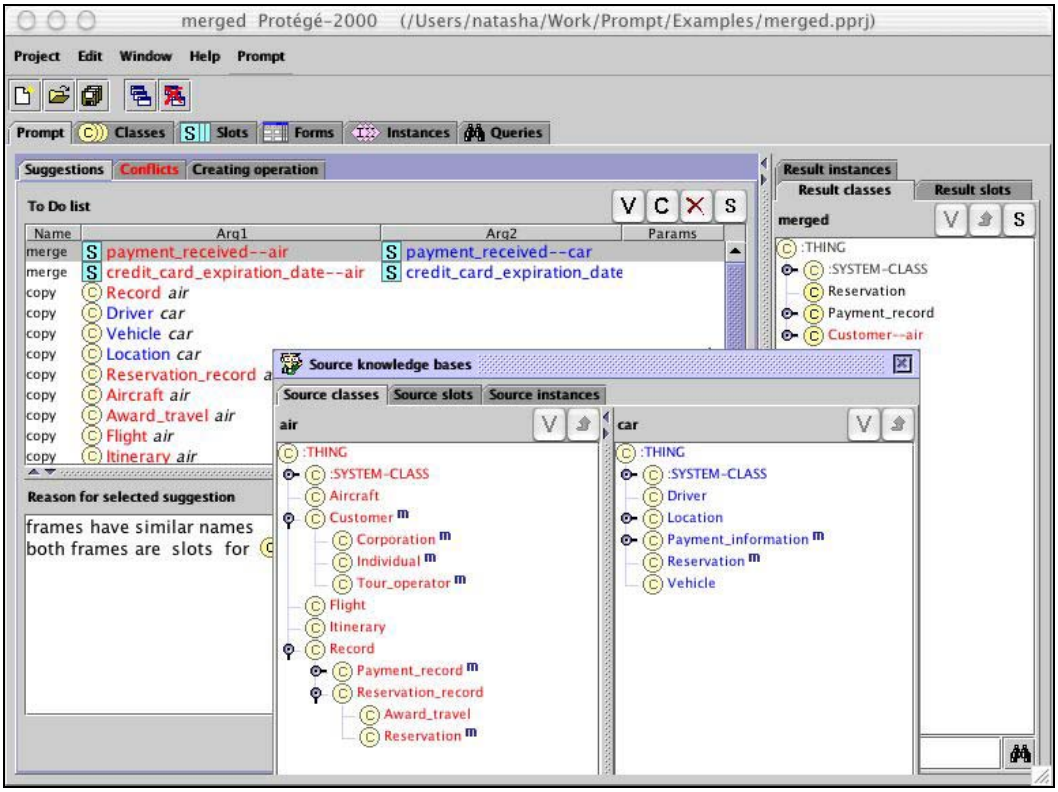

Figure 5.1. PROMPT screenshot. The main window (in the background) shows a list of current suggestions in the top left pane and the explanation for the selected suggestion at the bottom. The right-hand side of the window shows the evolving merged ontology. The internal screen presents the two source ontologies side-by-side (the superscript "m" marks the classes that have been merged or moved into the evolving merged ontology).

References

- Noy, N.F. and Musen, M.A. PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. In: Seventeenth National Conference on Artificial Intelligence (AAAI-2000). Austin, TX, 2000.
- Noy, N.F. and Musen, M.A. Anchor-PROMPT: Using Non-Local Context for Semantic Matching. In: Workshop on Ontologies and Information Sharing at the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-2001). Seattle, WA, 2001.

5.2.3 FCA-Merge – A Method for Bottom-Up Merging of Ontologies

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 they apply mathematically founded techniques taken from *Formal Concept Analysis* [Wille et al.; 1982] [Ganter et al.; 1999] to derive a lattice of concepts as a structural result of FCA-Merge. The produced result is explored and transformed to 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 is depicted in figure 5.2 and consists of three steps, namely (i) instance extraction and computing of two formal contexts K_1 and K_2, (ii) the FCA-Merge 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.

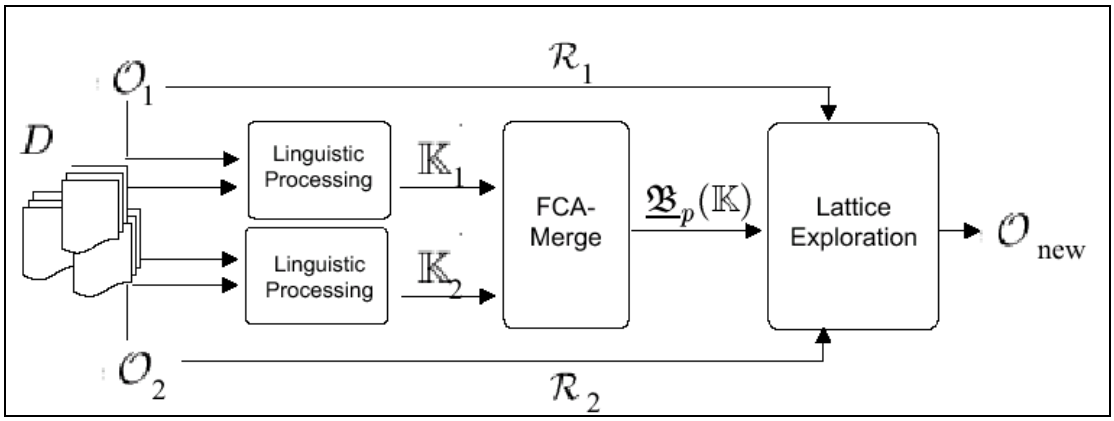

Figure 5.2. FCA-merge process

Instance Extraction

The extraction of instances from text documents circumvents 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 as input data the two ontologies and a set D of natural language documents. 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, we *extract instances*. This automatic knowledge acquisition step returns, for each ontology, a formal context 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 FCA Core Algorithm

The second step of our 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.

Deriving the Merged Ontology

Instance extraction and the FCA-Merge core algorithm are fully automatic. 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 R_1 and R_2, the ontology engineer creates the concepts and relations of the target ontology. We offer graphical means of the ontology-engineering environment OntoEdit for supporting this process.

Assumptions

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 our task. Secondly, the documents have to cover all concepts from the source ontologies. Concepts that are not covered have to be treated manually after our 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.

References

- B. Ganter, R. Wille: *Formal Concept Analysis: Mathematical foundations*. Springer, Berlin-Heidelberg 1999
- R. Wille: Restructuring lattice theory: an approach based on hierarchies of concepts. In: I. Rival (ed.): *Ordered sets*. Reidel, Dordrecht-Boston 1982, 445-470

5.2.4 Information-Flow-based Ontology Mapping

IF-Map [Kalfoglou and Schorlemmer 2002] is an automatic method for ontology mapping based on the Barwise-Seligman theory of information flow [Barwise and Seligman 1997]. A very brief description of this method is presented here because mappings have a strong relationship with ontology merging, and because IF-Map is partially inspired in FCA-merge.

The basic idea is to align two local ontologies by looking at how these are mapped from a common reference ontology. It is assumed that such reference ontology is not populated with instances, while local ontologies usually are. IF-Map generates possible mappings between an unpopulated reference ontology and a populated local ontology by taking into account how local communities classify instances with respect to their local ontologies. The mathematical foundations of the method are described in [Schorlemmer 2002].

References

- J. Barwise and J. Seligman. Information Flow: The Logic of Distributed Systems. Cambridge Tracts in Theoretical Computer Science 44. Cambridge University Press, 1997.
- Y. Kalfoglou and M. Schorlemmer. Information-Flow-based Ontology Mapping. In International Conference on Ontologies, Databases and Applications of Semantics. Irvine, California (USA), October 2002. To be published in Springer's LNCS series.
- M. Schorlemmer. Duality in Knowledge Sharing. In Seventh International Symposium on Artificial Intelligence and Mathematics. Fort Lauderdale, Florida (USA), January 2002.

5.2.5 The MOMIS methodology

MOMIS [Bergamaschi et al.; 1999] [Beneventano et al.; 2000] [Bergamaschi et al.; 2001] [Benetti et al.; 2002] (see figure 5.3) 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 schema.

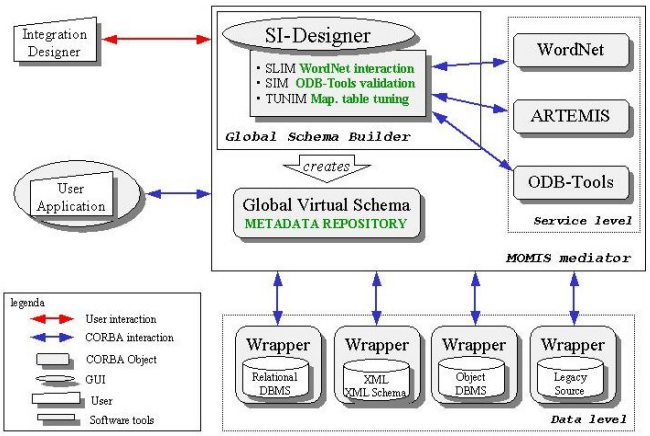

Figure 5.3. The MOMIS Architecture

The system architecture is composed of functional elements that communicate using the CORBA standard. A data model, ODM_{I3}, and a language, ODL_{I3} are used to describe information source. ODL_{I3} and ODM_{I3} have been defined as 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_{I3} representation. The Global Virtual Schema (GSB) module processes and integrates descriptions received from wrappers to derive the global shared schema by interacting with different service modules, namely ODB-Tools, an integrate environment for reasoning on object oriented database based on Description Logics [Beneventano et al.; 2002], WordNet lexical database that supports the mediator in building lexicon-derived relationships, and ARTEMIS tool that performs the clustering operation [Castano et al.; 2001].

The MOMIS integration process

In order to create a global virtual schema of involved sources, MOMIS generates a common thesaurus of terminological intensional and extensional relationships describing intra and inter-schema knowledge about classes and attributes of the source schemas. On the basis of the common thesaurus contents, MOMIS evaluates affinity between intra and inter-sources classes and groups similar classes together in clusters using hierarchical clustering techniques. A *global class*, that becomes representative of all the classes belonging to the cluster, is defined for each cluster. The global view for the involved source data consists of all the global classes. A graphical tool, the Source Integration Designer, SI-Designer, supports the MOMIS methodology.

Building the common thesaurus

The common thesaurus is built through an incremental process during which relationships are added in the following order: schema-derived relationships, lexicon-derived relationships, designer-supplied relationships and inferred relationships.

Schema-derived relationships. MOMIS extracts intensional relations from schemas structure knowledge, by analyzing each ODL_{I3} schema separately. For example, MOMIS extracts intra-schema RT relationships from the specification of foreign keys in relational source schema and from part-of relationship in hierarchical sources.

Lexicon-derived relationships. MOMIS extracts the lexical relationships by analyzing different source schemas, according to the WordNet (www.cogsci.princeton.edu/wn) ontology. For each element composing the schemas of the involved sources, the user has to choose the associated word in the WordNet system. This choice consists of choosing both a base form and a meaning. Our system tries to automatically suggest a base form. Starting from the base form and the meanings associated to each sources' element, the system inserts into the common thesaurus the lexicon-derived relationships obtained by exploiting the names properties stored in WordNet.

Designer-supplied relationships. The designer may supply further relationships to capture specific domain knowledge about the source schemas.

Checking consistency and inferring new relationships. In this step, MOMIS performs reasoning about the common thesaurus relationships by exploiting the subsumption and inheritance computation, reasoning techniques of Description Logics performed by ODB-Tool [Beneventano et al.; 1997].

Clustering ODL_{I3} classes

To integrate the ODL_{I3} classes of the different sources into a global ODL_{I3} classes, MOMIS employs hierarchical clustering techniques based on the concept of affinity. In this way, MOMIS identifies ODL_{I3} 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 [Castano et al.; 2001], evaluating a set of affinity coefficients for all possible pairs of ODL_{I3} 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, where the classes themselves are the leaves and intermediate nodes.

Global class and mapping tables

Starting from clusters generated at the previous stage, MOMIS defines a global class for each cluster that represents the mediated view of all the cluster's classes. For each global class, Momis provides a set of global attributes and, for each of these attributes, the mappings of the local attributes—i.e. the attributes of the local classes belonging to the cluster. Momis obtains the global attributes in two steps: union of the attributes of all the classes belonging to the cluster, and fusion of the similar attributes by using mapping functions. In the second step, Momis eliminates redundancies semi–automatically by taking into account the relationships stored in the common thesaurus. For each global class, Momis generates a persistent mapping table that stores all the mappings.

References

- S. Bergamaschi, S. Castano e M. Vincini "Semantic Integration of Semistructured and Structured Data Sources", SIGMOD Record Special Issue on Semantic Interoperability in Global Information, Vol. 28, No. 1, March 1999
- Beneventano D., Bergamaschi S., Castano S., Corni A., Guidetti R., Malvezzi G., Melchiori M., Vincini M., *Information Integration: the MOMIS Project Demonstration*, Proceedings of 26th Int.Conf.on Very Large Data Bases, 2000, Cairo, Egypt.
- Bergamaschi S., Castano S., Beneventano D., Vincini M., *Semantic Integration of Heterogeneous Information Sources*, Special Issue on Intelligent Information Integration, Data & Knowledge Engineering, Vol. 36, Num. 1, 215-249, Elsevier Science B.V. 2001.
- Benetti I., Beneventano D., Bergamaschi S., Guerra F., Vincini M., *An Information Integration Framework for e-commerce*, IEEE Intelligent Systems, (Jan/Feb 2002).
- Beneventano D., Bergamaschi S., Sartori C., Vincini M., *ODB-Tools: A Description Logics Based Tool For Schema Validation and Semantic Query Optimization in Object Oriented Databases*, Proc. of Int. Conf. on Data Engineering (ICDE-97), Birmingam UK 1997.
- Castano S, De Antonellis V., De Capitani Di Vimercati S, *Global Viewing of Heterogeneous Data Sources*, IEEE Transactions TKDE 13(2): 277-297 (2001).

6 Ontology evolution methods

Ontology evolution can be defined as timely adaptation of the ontology as well as the consistent propagation of changes. A modification in one part of the ontology may generate subtle inconsistencies in other parts of the same ontology, in the instances, dependent ontologies and applications. This variety of causes and consequences of the ontology changes, which is discussed in [Klein et al., 2002], makes ontology evolution a very complex operation that should be realized as an organizational process as presented in the Figure 6.1. The problem of schema evolution has been extensively studied in relational and database papers (see [Roddick, 1995] and [Franconi et al. 2000]. In [Noy et al, 2002], authors discuss the differences that steam from different knowledge models and different usage paradigms.

In the following, we will shortly elaborate on each of the phases of the ontology evolution process.

Representation. To resolve changes, they have to be identified and represented in a suitable format. Elementary changes in the ontology (e.g. add concept, remove property, set property range etc.) are derived from the conceptual model [Motik et al. 2002]. However, this granularity of ontology changes is not always appropriate. Often, the intent of the changes may be better expressed on a higher level. For example, the user may need to move a concept form one parent to another. She may bring the ontology into desired state through a successive application of a list of elementary evolution changes (i.e. "remove subConceptOf" and "add subConceptOf" elementary changes). However, a lot of unnecessary changes may be performed if each change is applied alone. To avoid that drawback, it should be possible to express changes on a more coarse level, with the intent of change directly visible. Therefore, we introduce the composite changes (e.g. move concept) representing a group of elementary changes applied together [Stojanovic et al. 2002].

Semantics of Change. Application of a change in the ontology can induce inconsistencies in other parts of the ontology. We distinguish semantic and syntactic inconsistency. Semantic inconsistency arises if the meaning of an ontology entity is changed. Syntactic inconsistency arises if undefined entities at the ontology or instance level are used, or if ontology model constraints are invalidated. For example, the removal of a concept which is the only element of a domain set for some property results in syntactic 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. If an ontology is large, it may be difficult to fully comprehend the extent and meaning of each induced change. The task of the '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 of effects of each change, this phase should contribute to the maximum transparency providing detailed insight into each change being performed.

For each change in the ontology, it is possible to generate a set of additional changes, leading to different final consistent states. Evolution strategies [Stojanovic et al., 2002] are mechanisms used in this phase allowing the user to customize the ontology evolution process according to her needs. Consequently, the user can transfer ontology in the desired consistent state.

Implementation. In order to avoid performing undesired changes, before applying a change to the ontology, a list of all implications to the ontology should be generated and presented to the user. She should be able to comprehend the list and approve or cancel the changes. When the changes are approved, they are performed by successively resolving changes from the list. Since it is necessary to perform several changes together, the transaction server is needed. If changes are cancelled, the ontology should remain intact.

Propagation. The task of the change propagation phase is to automatically bring all dependent elements to a consistent state after an ontology update has been performed. First, when the ontology is modified, ontology instances need to be changed to preserve consistency with the ontology. Second, an ontology update might also corrupt ontologies that depend on the modified ontology and, consequently, all artefacts that are based on these ontologies. This problem can be solved recursive by applying the ontology evolution process to these ontologies. However, apart from the syntax inconsistency, the semantic inconsistency can also arise when, for example, the dependent ontology already contains a concept that is added to the original ontology. Third, when an ontology is changed, applications based on the changed ontology may not work correctly anymore. An ontology evolution approach has to recognize which change in the ontology can affect the functionality of dependent applications and react correspondingly.

Validation. When working on an ontology collaboratively, different parties may have different ideas about how the ontology should be changed. Further, one party may fail to understand the actual effect of the change and approve the change that shouldn't be performed. Moreover, it may be desired to change the ontology for experimental purposes. In order to enable recovering from these situations, we introduce the 'validation phase' in the ontology evolution process. Validation concerns the truthfulness of an ontology with respect to its problem domain – does the ontology represent a piece of reality and the users' requirements correctly? It enables the acknowledgement of performed changes and undoing them at the user's request. 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 deleted from a concept hierarchy, its subconcepts will need to be either deleted as well, attached to the root concept, or attached to the parent of the deleted concept. Reversing such a change is not equal to recreating the deleted concept – one also needs to revert the concept hierarchy to the original state. The problem of reversibility is typically solved by creating evolution logs. An evolution log tracks information about each change, allowing the reconstruction of the sequence of changes leading to the original state of the ontology.

Discovery. The previously described 'validation phase' results in an ontology which, although in a consistent state, may contain redundant entities or may be better structured with respect to the domain. For example, multiple users may be working on different parts of an ontology without enough communication. They may be deleting subconcepts of a common concept at different points in time to fulfill their immediate needs. As a result, it may happen that only one subconcept is left. Since classification with only one subclass beats the original purpose of classification, we consider such ontology to have a suboptimal structure. To aid users in detecting such situations, we investigated the possibilities of applying the self-adaptive system principles and proactively making suggestions for *ontology refinements* – changes to the ontology with the goal of improving the ontology, making the ontology easier to understand and cheaper to modify. We have identified following ways of discovering changes [Stojanovic et al., 2002]:

- *Structure-driven change discovery* exploits a set of 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;
- *Data-driven change discovery* detects the changes which are induced through the analysis of existing instances. For example, if no instance of a concept C uses any of the properties defined for C, but only properties inherited from the parent concept, we can make an assumption that C is not necessary;
- *Usage-driven change discovery* 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

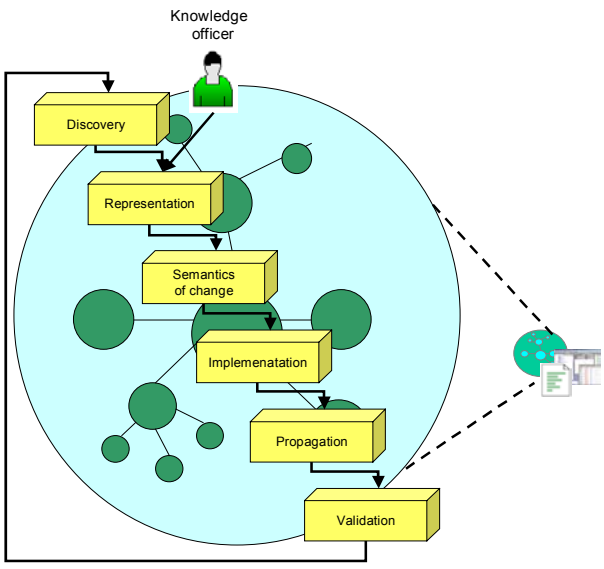

Figure 6.1. The Ontology Evolution Process

References

- E. Franconi, F. Grandi, and F. Mandreoli, "A semantic approach for schema evolution and versioning in object-oriented databases", Proc. CL2000, 2000.
- M. Klein, A. Kiryakov, D. Ognyanov and D. Fensel. "Ontology Versioning and Change Detection on the Web". Proceedings of the 13th European Conference on Knowledge Engineering and Management, EKAW-2002, Springer, LNAI, Madrid, Spain, 2002.

- B. Motik, A. Maedche and R. Volz: "A Conceptual Modeling Approach for building semantics-driven enterprise applications". Proceedings of the First International Conference on Ontologies, Databases and Application of Semantics (ODBASE-2002), Springer, LNAI, California, USA, 2002.
- N. F. Noy, M. Klein: "Ontology Evolution: Not the Same as Schema Evolution", SMI technical report SMI-2002-0926, 2002.
- J.F. Roddick: "A Survey of Schema Versioning Issues for Database Systems", Information and Software Technology, 37(7):383-393, 1995.
- L. Stojanovic, A. Maedche, B. Motik and N. Stojanovic: "User-Driven Ontology Evolution Management". Proceedings of the 13th European Conference on Knowledge Engineering and Management, EKAW-2002, Springer, LNAI, Madrid, Spain, 2002.
- L. Stojanovic, N. Stojanovic, A. Maedche: "Change Discovery in Ontology-based Knowledge Management Systems". Second International Workshop on Evolution and Change in Data Management, held in conjunction with the 21st International Conference on Conceptual Modelling, ER 2002, Finland, 2002.

7 Ontology evaluation methods

7.1 Introduction

Currently the semantic web attracts researchers from all around the world. Numerous tools and applications of semantic web technologies are already available and the number is growing fast. Ontologies play an important role for the semantic web as a source of formally defined terms for communication. They aim at capturing domain knowledge in a generic way and provide a commonly agreed understanding of a domain, which may be reused, shared, and operationalised across applications and groups. The large visibility of the semantic web, its tools and applications already attracts industrial partners, e.g. in numerous projects funded by the European Commission. As they move from academic institutions into commercial environments they have to fulfil stronger requirements (e.g. concerning correctness, consistency, completeness, conciseness, etc.). Therefore, evaluation is a key activity in ontology development.

7.2 Description of methods for ontology evaluation

This section presents, in chronological order, the most well known approaches for building ontologies both from scratch, or reusing ontologies as they are on the ontology libraries. We will provide a brief description of each methodology, presenting who has elaborated it, the proposed steps, the ontologies that have been developed using the methodology, and the applications where such ontologies have been used. We also provide references for each methodology.

7.2.1 Gómez Pérez's approach for taxonomy evaluation

Most of the existing ontologies are formalized under a frame-based approach and domain knowledge is structured in taxonomies. In [Gómez-Pérez, 01], you can find a set of errors that can be made when building taxonomies. This study is based on the experience of evaluating ontologies at the Ontolingua Server, and should be used to prevent this kind of errors occurring when developing new ontologies. Throughout this section, we will use the semantics of the primitives defined at the frame ontology [Gruber, 93].

The Frame Ontology Primitives

The Ontolingua Server has a library of ontologies written in Ontolingua. Ontolingua ontologies were originally written in KIF [Genesereth et al.; 92]. A knowledge representation ontology, named the Frame Ontology, was built on KIF. Its goal was to unify the semantics of the most commonly used primitives in the frame paradigm and enable future ontology developers to develop ontologies using a frame-based approach. When building a taxonomy using the frame ontology vocabulary, the following primitives are used:

- *Subclass-of (?child-class ?parent-class)*: the class ?child-class is a subclass of the class ?parent-class
- *Superclass-of (?parent-class ?child-class)*: the class ?parent-class is a superclass of the class ?child-class.
- *Instance-of (?individual ?class):* the instance ?individual is an instance of the class ?class.
- *Class-Partition (?set-of-classes):* defines a set of disjoint classes. A partition of the class C into the set of classes class_p_1, ..., class_p_n, where class_p_i ≠ class _p_k for every i ≠ k is defined when any instance or subclass of any class class_p_i cannot belong to any other class class-p_k. When this occurs, the classes class_p_1, ..., class_p_n are a partition of class C.
- *Subclass-Partition (?C ?Class-partition*): defines the set of disjoint subclasses ?Class-Partition as subclasses of the class ?C. This classification does not necessarily have to be complete, that is, there may be instances of ?C that are not included in any of these subclasses.
- *Exhaustive-Subclass-Partition(?C ?Class-Partition*): defines the set of disjoint subclasses ?Class-Partition as subclasses of the class ?C, where the class ?C can be defined as a union of all the classes that make up the partition.

Partitions can define concept classifications in a disjoint and/or complete manner. As exhaustive subclass partitions merely add the completeness constraint to the established subsets, they have been distinguished as: non-exhaustive subclass partition errors and exhaustive subclass partition errors.

Errors in developing taxonomies

Gómez-Pérez presents a set of possible errors that can be made by ontologists when building taxonomic knowledge into an ontology or by Knowledge Engineers when building KBs under a frame-based approach. They are classed under the following criteria: inconsistency errors, incompleteness errors and redundancy errors.

A) Inconsistency Errors

Circularity errors. They occur when a class is defined as a specialization or generalization of itself. Depending on the number of relations involved, circularity errors can be classed as: circularity errors at distance zero (a class with itself), circularity errors at distance 1 and circularity errors at distance n.

Partition Errors. Partitions can define concept classifications in a disjoint and/or complete manner. As exhaustive subclass partitions merely add the completeness constraint to the established subsets, subclass partition errors and exhaustive subclass partition errors have been distinguished:

- *Subclass partition with common instances***.** It occurs when one or several instances belong to more than one subclass of the defined partition. For example, if *dogs* and *cats* form a subclass partition of the set of *mammals*, an error of this type would occur if we define *Pluto* as an instance of both classes. The developer should remove the wrong relation to solve this problem.

- *Subclass partition with common classes* occurs when there is a partition $class\_p_1,\ldots, class\_p_n$ defined in a class *class_A* and one or more classes $class\_B_1, \ldots, class\_B_k$ are subclasses of more than one subclass $class\_p_i$ of the partition. For example, if *dogs* and *cats* form a class partition of the set of *mammals*, an error of this type would occur if we define the class *Doberman* as a subclass of both classes. The developer should remove the wrong relation to solve the problem.

- *Exhaustive subclass partition with common instances***.** It occurs when one or several instances belong to more than one subclass of the defined exhaustive partition. For example, having defined the classes *odd* and *even* as an exhaustive subclass partition of the class *number,* an error of this type appears if the number *four* is an instance of the *odd* and *even* numbers.

- *Exhaustive subclass partition with common classes* occurs when there is a partition $class\_p_1,\ldots, class\_p_n$ defined in a class *class_A* and one or more classes $class\_B_1, \ldots, class\_B_k$ are subclasses of more than one subclass $class\_p_i$ of the partition. For example, having defined the classes *odd* and *even* as an exhaustive subclass partition of the class *number,* an error of this type appears if the class *prime* is a subclass of the *odd* and *even* numbers. So, if we define the number *Three* as a prime number, we get the inconsistency since three would be an instance of odd and even numbers.

- *Exhaustive subclass partition with external instances.* These errors occur when having defined an exhaustive subclass partition of the base class (Class_A) into the set of classes class-p_1 ... class-p_n, there are one or more instances of the class_A that do not belong to any class class_p_i of the exhaustive partition. For example, if the *numbers* classed as *odd* and *even* had been defined as forming an exhaustive subclass partition and the number four were defined as an instance of the class *numbers* (instead of the class *even*), we would have an error of this type.

Semantic Inconsistency Errors. They usually occur because the developer makes an incorrect semantic classification, that is, classes a concept as a subclass of a class of a concept to which it does not really belong; for example, classes the concept *dog* as a subclass of the concept *house*. The same would occur with the instances; for example, if the instance *Pluto* would be an instance-of of the class *house.*

B) Incompleteness Errors

Errors of this type are classed as:

Incomplete Concept Classification. Generally, an error of this type is made whenever concepts are classified without accounting for them all, that is, concepts existing in the domain are overlooked. An

error of this type occurs if a concept classification *musical instruments* is defined considering only the classes formed by *string instruments* and *wind instruments* and overlooking, for example, the *percussion instruments*.

Partition Errors. They could appear when the definition of the partition between a set of classes is omitted. We have identified two types of errors:

- *Subclass partition omission.* The developer identifies the set of subclasses of a given class, but omits that the subclasses are disjoint. An example would be to define *dogs* and *cats* as a subclass of *mammals* and to omit that *dogs* and *cats* form a subclass partition (though not complete) of the set of *mammals*.

- *Exhaustive subclass partition omission*. The developer defines a partition of a class and omits the completeness constraint to the established subsets. Examples would be to define *odd* and *even* as a subclass of *numbers* or to define *odd* and *even* as a subclass partition of *numbers.* In both cases, it is omitted that the *numbers* classed as *odd* and *even* form an exhaustive subclass partition (that is, complete).

C) Redundancy Errors.

Redundancy is a type of error that occurs when redefining expressions that were already explicitly defined or that can be inferred using other definitions.

Gramatical Redundacy Errors. These errors occur in taxonomies when there is more than one explicit definition of any of the hierarchical relations.

- *Redundancies of subclass-of relations* occur between classes when subclass-of relations are repeated. We can distinguish direct and indirect repetition. It exists a *Direct repetition*, when two or more subclass of relations between the same source and target classes are defined, that is, including the subclass of relation between the classes *dog* and *mammals* twice. It exists a *Indirect repetition,* for example, if we define the class *dog* as a subclass of *pet,* and *pet* as a subclass of *animal,* when *dog* is also defined as a subclass of *animal.*

- *Redundancies of instance-of relations.* As in the above case, there are two possibilities. It exists a *direct repetition* if two *instance-of* relations between the same instance and class are defined. It exists an indirect repetition, for example, if we define the instance *Clyde* as an instance of *real elephant* and *real elephant* as a subclass of the class *elephant.* The definition of an instance of relation between *Clyde* and *elephant* would lead to a redundancy in the taxonomy.

Identical formal definition of some classes. It occurs when there are two or more classes in the ontology with the same formal definition, that is, the only difference between the subclasses is the name. This error type could be also be classified as an example of classes with incomplete knowledge. The developer could solve this problem by adding what distinguishes the classes of the partition or, otherwise, realize that it does not make sense to have classes with identical formal definitions in the partition and delete one of them.

Identical formal definition of some instances. It occurs when there are two or more instances in the ontology with the same formal definition, that is, the only difference between them is the name.

References

- Genesereth M., Fikes R.. (1992). Knowledge Interchange Format, Technical Report, Computer Science Department, Stanford University, Logic-92-1.

- Gómez-Pérez, A. “Evaluation of Ontologies”. *International Journal of Intelligent Systems. 16(3)*. March, 2001.

- Gómez-Pérez, A. "Evaluation of Taxonomic Knowledge in Ontologies and Knowledge Bases". *Twelfth Workshop on Knowledge Acquisition, Modeling and Management (KAW'99)*. Alberta (Canada). 1999.

- Gómez-Pérez, A. *A Framework to Verify Knowledge Sharing Technology.* Expert Systems with Application. Vol. 11, No. 4. pp. 519-529. 1996.

- Gómez-Pérez, A.; Juristo, N.; Pazos, J. *Evaluation and assessment of knowledge sharing technology*. In N. J. Mars (editor), Towards Very Large Knowledge Bases. KBKS 95. IOS Press, Amsterdam, 1995. pp., 289-296.
- Gómez-Pérez, A. (1994a).Some ideas and Examples to Evaluate Ontologies. Technical Report KSL-94-65. Knowledge System Laboratory. Stanford University. Also in Proceedings of the 11th Conference on Artificial Intelligence for Applications. CAIA94.
- Gómez-Pérez, A. (1994b). From Knowledge Based Systems to Knowledge Sharing Technology: Evaluation and Assessment. Technical Report. KSL-94-73. Knowledge Systems Laboratory. Stanford University. December 1994.

7.2.2 Ontological Constraints Manager (OCM)

In the context of a EU funded TMR fellowship, Kalfoglou and colleagues investigated the role of ontological axioms in engineering. In particular, they point out that the role anticipated by ontological axioms is rarely delivered: to restrict the possible interpretations ontological constructs could have.

Ontological axioms

For example, the textual form of an ontological axiom of a formal ontology, the Process Interchange Format (PIF) ontology [Lee98] is as follows: *"An object can participate_in an activity only at those time points at which both the object exists and the activity is occurring."*. This axiom can be formalised in first order logic as follows:

$participates\_in(O,A,T) \rightarrow exists\_at(O,T) \wedge is\_occurring\_at(A,T)$.

where O,A and T are variables used for object, activity and time points, respectively. The role of this axiom is to restrict possible interpretations of the ontologically defined relation *participates_in/3*. Hence the right-hand side implication of the formula above: if the right part of the implication is true, then the left part has to be true. This constraining role of the axioms makes it possible to view them as ontological constraints. So, whenever someone using the PIF ontology describes the relation in a way that does not conform to the axiomatised definition this will reveal a potential discrepancy. For instance, the following definition, which might be part of a logic specification using the ontology (hence, the left-hand side implication):

$participates\_in(O,A,T) \leftarrow exists\_at(O,T) \wedge performs(O,A)$.

could be erroneous with respect to the axiom given above, depending on how *is_occurring_at/2* is defined for activities. However, it uses valid ontological constructs (assuming that *performs/2* relation is also defined in the ontology), which makes it difficult to detect. It reflects a misunderstanding of ontology's semantics (in this case, proper use of *participates_in/3* relation) and can only be detected by checking its conformity with the existing axiomatisation.

Conformance check

To operationalise this idea and enforce the role of ontological constraints in checking the conformance of specifications to underlying ontologies, Kalfoglou and colleagues developed an architecture where ontological constraints are deployed for consistency checking. In particular, they invented an architecture in which ontological constraints are separated from other ontological constructs. These are enforced to comply with the axiomatisation in order to verify the consistency of the specification with respect to domain knowledge as it is explicitly represented in the underlying ontology. For the sake of brevity we do not describe in detail the technicalities concerned with the use of this architecture but we point the interested reader to [Kalfoglou99] for an in-depth discussion.

This use of ontological constraints has two clear benefits for systems design:

- *augmentation* of formal specifications with formally defined constructs drawn from the underpinning ontology. We argue that these constructs might be reused in other similar applications which may result in a cost-effective solution for the design process. (see, for example, [Kalfoglou00]);

- use of ontological axioms to detect *conceptual errors* in specifications that use ontological constructs. This has a potential impact to the early phases of software design since it makes it possible to detect errors that were previously uncaught. Moreover, these ontological constraints might be reused to detect similar kind of errors in other applications.

Multi-layer architecture

The assumption made in this way of using ontological constraints is that these are specified correctly. However, if ontological constraints are erroneous (and consequently, the ontology they come from, incorrect), it could lead to an erroneous error diagnosis. To tackle this problem we extended the architecture in such a way to make it possible to check many specifications and their ontological constraints simultaneously. This multi-layer architecture is now composed of an arbitrary number of layers each one representing a specification/ontology pair: assume that at the lower layer a specifier constructs the specification by using constructs from the chosen ontology. The specification should also conform to the ontological constraints provided by the ontology. This can be checked with our error checking mechanism, as we already described above.

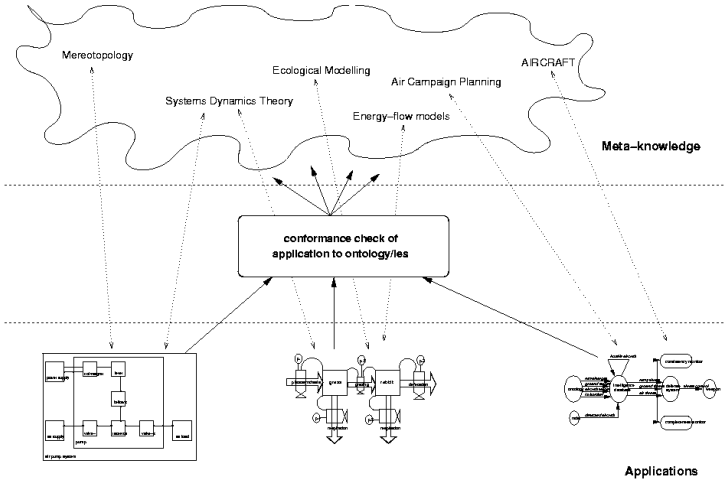

Figure 7.1. The multi-layer approach: it enforces the conformance check of an application to ontology/ies. Applications are using meta-knowledge constructs from various ontologies (mereotopology, system dynamics theory, etc.) in an integrated environment that enables checks on the use of those constructs against their axiomatised definitions.

However, if an ontological constraint has been erroneously defined we can check this for error with our flexible mechanism. This is possible because we treat ontological constraints as a specification in a layer above the one to which we applied the same checks as for specifications. Ontological constraints are now checked for errors against another set of constraints which can be viewed as meta-level constraints. They are part of the ontology and their use is to verify the correctness of the constraints. The result of this check will be the detection of an error, if any, in the ontological constraints. Ultimately, this layer checking can be extended to an arbitrary number of layers upwards, until no more layers can be defined. Although the multi-layer architecture looks similar to the traditional view of modularity in software development it differs in the sense that each new layer (or module) that is added to the indexed lattice is used to verify the correctness of the layer beneath it. Consequently, another layer can be added at the top of it to check its correctness. Hence, there is no restriction as to how many layers can be defined in the architecture. The details of this multi-layer architecture are given in [Kalfoglou99c], and an illustrative diagram is included in the figure 7.1.

The advantage is that we can capture a wide variety of errors occurring at different layers. It is possible to view the constraints introduced at each layer of error checking as an ontology and to check these for each query of the program using the same mechanism. This implies that the ontology in each layer need not be the same. We applied this approach to a set of ontologies where each layer represents a different ontology. Assuming that the ontologies placed in the architecture are somehow related (for example, via an inclusion lattice), we can check for correctness an ontology against another one. In figure 7.2 we illustrate a descriptive example of the multi-layer checking in the PhysSys ontologies set [Borst97]. In

the left part of the figure we include the identification number of a layer, 0 for the PhysSys ontology, 1 for Process ontology and so on. In the middle we illustrate the detection of erroneous definitions of mechanicm and disjoint starting checking from layer 0. The dashed lines starting from the body of relation conn2ef/2 denote the path being followed in order to detect the errors. Those occurred in layers 1 (Process ontology) and 5 (Mereology ontology) for definitions mechanism and disjoint, respectively. The right part shows which ontologies are included in the error check per layer. PhysSys is composed of 7 different ontologies, and represents the domain of system dynamics for physical systems.

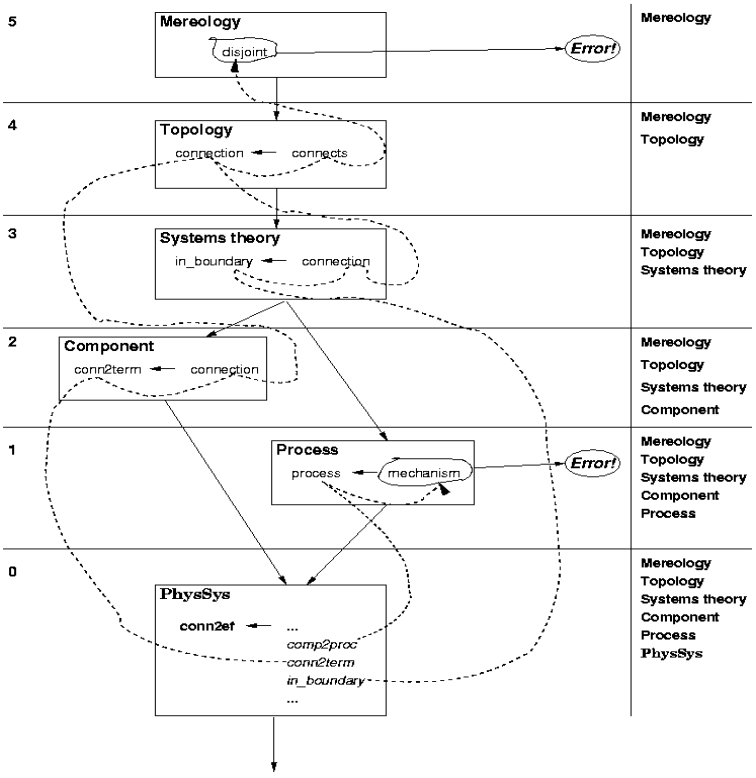

Figure 7.2. Detection of two errors occurred during the execution of the *conn2ef/2* relation at the top level ontology, **PhysSys**.

Implementation

The Java-written tool Ontological Constraints Manager (OCM) realise the multi-layer architecture described above. The underlying mechanized inference is provided by a Prolog engine which is linked to the Java front-end using a bi-directional interface provided by SICStus Prolog vendor. The whole system is described in [Kalfoglou99d].

References:

- Borst,P., Akkerman,H., Top,J., *"Engineering Ontologies"*, International Journal of Human-Computer Studies, (46)365-406, 1997.
- Kalfoglou,Y., Robertson,D., *"Use of Formal Ontologies to Support Error Checking in Specifications"*, in Proceedings of the 11th European Workshop on Knowledge Acquisition, Modelling and Management (EKAW99), Dagsthul, Germany, May 1999.
- Kalfoglou,Y., Robertson,D., *"A case study in applying ontologies to augment and reason about the correctness of specifications"*, in Proceedings of the 11th International Conference on Software Engineering and Knowledge Engineering (SEKE99), Kaiserslautern, Germany, June 1999.
- Kalfoglou,Y., Robertson,D., *"Managing Ontological Constraints"*, in Proceedings of the IJCAI99 workshop on Ontologies and Problem-Solving Methods (KRR-5), Stockholm, Sweden, August 1999.
- Kalfoglou,Y., Robertson,D., Tate,A., "Using Meta-Knowledge at the application level", University of Edinburgh, Dept. of AI, Research Paper No.956, September 1999.

- Lee,J., Gruninger,M. Jin,Y., Malone,T., Tate,A., Yost,G. and other members of the PIF working group. *"The PIF Process Interchange Format and framework"*, Knowledge Engineering Review, 13(1):91-120, February 1998.

7.2.3 OntoClean

OntoClean has been elaborated by the Ontology Group of the LADSEB-CNR in Padova (Italy). It is a method to clean taxonomies according to notions such as: *rigidity*, *identity* and *unity*. Let us see these notions [Gangemi et al.; 2001]:

- *Rigidity*. This notion is defined based on the idea of essence. A property is essential to an individual if and only if necessarily holds for that individual. Thus, a property is *rigid* (+R) if and only if is necessarily essential to all its instances. A property is *non-rigid* (-R) if and only if it is not essential to some of its instances, and a*nti-rigid* (~R) if and only if it is not essential to all its instances. For example, the concept `person` is usually considered rigid, since every person is essentially such, while the concept `student` is not normally considered anti-rigid, since every student can possibly be a non-student a few years later.
- *Identity*. A property *carries an identity criterion* (IC) (+I) if and only if all its instances can be (re)identified by means of a suitable "sameness" relation. A property *supplies an identity criterion* (+O) if and only if such criterion is not inherited by any subsuming property. For example, *person* is usually considered a supplier of an identity criterion (for example the fingerprint), while `student` just inherits the identity criterion of `person`, without supplying any further identity criteria.
- *Dependency*. An individual *x* is constantly dependent on *y* if and only if, at any time, *x* cannot be present unless *y* is fully present, and *y* is not part of *x*. For example, a hole in a wall is constantly dependent on the wall. The hole cannot be present if the wall is not present. A property *P* is constantly dependent if and only if, for all its instances, there exists something they are constantly dependent on. For instance, the concept hole is constantly dependent because every instance of hole is constantly dependent.
- *Unity*. We can say that an individual is a *whole* if and only if it is made by a set of parts unified by a relation *R*. For example, the enterprise Iberia is a whole because it is composed by a set of people that are linked by the relation having the same president. A property *P* is said to *carry unity* (+U) if there is a *common* unifying relation R such that all the instances of *P* are wholes under *R*[6]. For example, the concept enterprise-with-president carries unity because every enterprise with president is made up people linked through the relation having the same president. A property carries *anti-unity* (~U) if all its instances can possibly be non-wholes. Properties that refer to amounts of matter, like *gold, water,* etc., are good examples of anti-unity.

Note that the definition of these notions refer to properties of properties. For example, *rigid* is a property that can take different values in different properties (*yes* in `person`, *no* in `student`, etc.). Another example is *carries an identity criterion*, since it can also take different values in different properties (*yes* in `person`, *no* in `student`, etc.). These properties of properties are called **meta-properties**, and to indicate their values, special symbols are used. For example, +R means that the meta-property *rigid* has the value *yes*. The meta-properties are useful to detect wrong *subclass of* relations. For example, `person` cannot be subclass of `student` because the former one is rigid and the later one not. In fact, if we had this link, what would it happen if a person was not student any more?

According to LADSEB-CNR's proposal, the specific steps to clean the wrong *subclass of* links in a taxonomy are (based on [Welty et al.; 2001] and interviews with LADSEB-CNR's group):

1) *Put tags to every property assigning meta-properties*. This eases the analysis, because all the meta-properties are simultaneously visible.
2) *Focus just on the rigid properties*. A taxonomy without rigid properties is called *backbone taxonomy*. It is the base of the rest of the taxonomy, that is, the essential part.
3) *Evaluate the taxonomy taking into account principles based on the meta-properties*. For instance, a rule suggested in OntoClean is "a property carrying anti-unity has to be disjoint of a property carrying unity". As a consequence, "a property carrying unity cannot be a subclass of a property carrying anti-unity". Therefore, `bronze statue` (it carries unity) cannot be a subclass of `bronze` (it carries anti-unity), for example.

[6] In the actual definition, the authors use *essential wholes* instead of *wholes*. We have used just *wholes* because of clarity reasons.

4) *Consider non-rigid properties*. When the backbone taxonomy has been examined, the modeller has to evaluate the non-rigid properties. One of the proposed rules is: "a rigid property and an anti-rigid property are ever disjoint". As a consequence, "a non anti-rigid property cannot be a subclass of an anti-rigid property". Therefore, `person` (rigid) cannot be a subclass of `student` (anti-rigid).
5) *Complete the taxonomy with other concepts and relations*. There can be several reasons to introduce new concepts. One of them is the transformation of concepts in relations, for example, `student` could be transformed into a relation between `person` and `university`.

OntoClean has been used by IBM, OntologyWorks[7], Document Development Corporation[8]. At the Italian National Research Council Laboratories (LADSEB-CNR and ITBM-CNR), in Padova and Rome, OntoClean is in use in several projects including the development of an upper-level ontology based on a restructuring of WordNet, and the development of a core ontology for financial knowledge interchange [Guarino et al.; 2002].

References

- Gangemi, A., Guarino, N., Masolo, C., and Oltramari, A. 2001. Understanding top-level ontological distinctions. Proc. of IJCAI 2001 workshop on Ontologies and Information Sharing .
- Welty, C.; Guarino, N. *Supporting Ontological Analysis of Taxonomic Relationships*. Data and Knowledge Engineering. September 2001.
- Guarino, N. and Welty, C. 2002. "Evaluating Ontological Decisions with OntoClean". *Communications of the ACM*, 45(2): 61-65.

[7] www.ontologyworks.com
[8] www.docdev.com

8 Conclusions

In this section we provide conclusions about all the approaches presented in the document. Therefore, this section is structured in the same way than the deliverable:

- **Methodologies and methods for building ontologies from scratch or reusing other ontologies *(section 2)***. The main conclusion, taken from the analysis of tables 2.1 to 2.4, is that *none of the approaches cover all the processes involved in ontology building, which were presented in section 2.2*. However, the following scale can be established between the methodologies and methods presented, from more complete to less complete.

  i. METHONTOLOGY is the approach that provides most detailed descriptions of each process.

  ii. However, the On-To-Knowledge methodology identifies more processes than the rest of approaches.

  iii. The main strength of Grüninger and Fox's methodology is its high degree of formality. However, the ontology life cycle is not completely specified. Furthermore, although several ontologies have been developed with this method, and there are also applications that use these ontologies, the methodology has only been tested on the business domain.

  v. Uschold and King's method has the same omissions as the above method and it is less detailed.

  vi. SENSUS-based method, which, apart from the shortcomings of the above approaches, does not mention the life cycle.

  vii. Bernaras et al.'s method, which, apart from the above omissions, has not been used to build many ontologies and applications.

  In summary, most of the approaches are focused on the development activities, specially on the the ontology implementation, and they do not pay too much attention to other important aspects related to management, evolution and evaluation of ontologies, which are really important on the context of the semantic web. This is due to the fact that the ontological engineering field is relatively new. However, *a low compliance with the criteria established in section 2.1 does not mean a low quality of the methodology or method*. As de Hoog (1998) states, a not very specified method can be very useful for an experienced group.

- **Ontology reengineering methods *(section 3)***. The method presented in section 3 is just a sound initial approach to carrying out the above-mentioned process, although it must be improved in later studies using more complex ontologies. In order to increase the reusability of the ontology to be reengineered, guidelines and criteria to achieve a higher degree of reusability are needed in the restructuring process. Another open issue is related to the relationship between the ontology that is being reengineered and existing top-level ontologies.

- **Cooperative construction of ontologies *(section 4)***. Co4 is a protocol for collaborative construction of consensual knowledge bases in the web. The method used at the (KA)2 initiative also proved to be useful, and used two kinds of people in the ontology construction process: ontopic agents and ontology coordinating agents.

- **Ontology merging methods and methodologies *(section 5)***. The strategies for merging ontologies are quite diverse: hierarchical clustering techniques, formal concept analysis, and analysis of terminological (i.e., synonymy and polysemy) relations and properties defined in concepts. All the methods presented in this section are able to merge concept taxonomies, attributes and relations. However, they should include new methods for merging other ontology components, such as axioms.

- **Ontology evolution approaches *(section 6)***. Ontologies are dynamic entities that evolve over time. The management of ontology evolution and the relationships between different versions of the same ontology are crucial problems to be solved. In this section we have summarized the work done in this area so far, and have identified the need to create and integrate robust methods inside the current methodologies to distinguish and recognize versions, with procedures for updates and changes in ontologies as well its technological support. We have also included some guidelines for performing this activity.

- **Ontology evaluation methods *(section 7)***. The field of ontology evaluation is just emerging. From the methodological perspective, evaluation activities should be introduced in more detail into ontology development methodologies. In this section, we have presented three approaches for ontology evaluation: Gómez-Pérez's approach, which is mainly concerned with the analysis of wrong taxonomic relationships in ontologies, and has been used to evaluate ontologies in the Ontolingua Server ontology library; Ontological Constraints Manager (OCM), which is focused on applying ontological axioms for evaluation; and OntoClean, which allows cleaning tangled concept taxonomies according to some philosophy-based notions (rigidity, identity and unity).

References

- De Hoog, R. "Methodologies for Building Knowledge Based Systems: Achievements and Prospects". In J. Liebowitz (Editor) *Handbook of Expert Systems*. CRC. 1998.