
D2.1.3.1 Report on Modularization of Ontologies

**Coordinated by Stefano Spaccapietra
(Ecole Polytechnique Fédérale de Lausanne)**

With contributions from:

**Maarten Menken, Heiner Stuckenschmidt, Holger Wache (Vrije
Universiteit Amsterdam)**

Luciano Serafini (Centro per la Ricerca Scientifica e Tecnologica, Trento)

Andrei Tamin (Università degli Studi di Trento)

Mustafa Jarrar (Vrije Universiteit Brussel)

Fabio Porto (Ecole Polytechnique Fédérale de Lausanne)

Christine Parent (Université de Lausanne)

Alan Rector, Jeff Pan (University of Manchester)

**Mathieu d'Aquin, Jean Lieber, Amedeo Napoli (Institut National de Recherche en
Informatique et en Automatique, Lorraine)**

**Giorgos Stoilos, Vassilis Tzouvaras, Giorgos Stamou (Centre for Research and
Technology Hellas, Informatics and Telematics Institute)**

Abstract.

EU-IST Network of Excellence (NoE) IST-2004-507482 KWEB

Deliverable D2.1.3.1 (WP2.1)

This deliverable gives an overview of concepts and methods necessary for achieving scalability through modularization of ontologies. This includes partitioning algorithms for large ontologies into smaller modules, distributed reasoning, e.g. distributed RDF querying, engineering approaches to ontology modularization, as well as an insight into composition (the inverse to modularization).

Keyword list: state-of-the-art, scalability, modularization, distribution, composition.

Document Identifier	KWEB/2004/D2.1.3.1/v1.1
Project	KWEB EU-IST-2004-507482
Version	v1.1
Date	July 30, 2005
State	final
Distribution	public

Knowledge Web 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-2004-507482.

University of Innsbruck (UIBK) - Coordinator

Institute of Computer Science
Technikerstrasse 13
A-6020 Innsbruck
Austria
Contact person: Dieter Fensel
E-mail address: dieter.fensel@uibk.ac.at

France Telecom (FT)

4 Rue du Clos Courtel
35512 Cesson Sévigné
France. PO Box 91226
Contact person : Alain Leger
E-mail address: alain.leger@rd.francetelecom.com

Free University of Bozen-Bolzano (FUB)

Piazza Domenicani 3
39100 Bolzano
Italy
Contact person: Enrico Franconi
E-mail address: franconi@inf.unibz.it

Centre for Research and Technology Hellas / Informatics and Telematics Institute (ITI-CERTH)

1st km Thermi - Panorama road
57001 Thermi-Thessaloniki
Greece. Po Box 361
Contact person: Michael G. Strintzis
E-mail address: strintzi@iti.gr

National University of Ireland Galway (NUIG)

National University of Ireland
Science and Technology Building
University Road
Galway
Ireland
Contact person: Christoph Bussler
E-mail address: chris.bussler@deri.ie

École Polytechnique Fédérale de Lausanne (EPFL)

Computer Science Department
Swiss Federal Institute of Technology
IN (Ecublens), CH-1015 Lausanne
Switzerland
Contact person: Boi Faltings
E-mail address: boi.faltings@epfl.ch

Freie Universität Berlin (FU Berlin)

Takustrasse 9
14195 Berlin
Germany
Contact person: Robert Tolksdorf
E-mail address: tolk@inf.fu-berlin.de

Institut National de Recherche en Informatique et en Automatique (INRIA)

ZIRST - 655 avenue de l'Europe -
Montbonnot Saint Martin
38334 Saint-Ismier
France
Contact person: Jérôme Euzenat
E-mail address: Jerome.Euzenat@inrialpes.fr

Learning Lab Lower Saxony (L3S)

Expo Plaza 1
30539 Hannover
Germany
Contact person: Wolfgang Nejdl
E-mail address: nejdl@learninglab.de

The Open University (OU)

Knowledge Media Institute
The Open University
Milton Keynes, MK7 6AA
United Kingdom
Contact person: Enrico Motta
E-mail address: e.motta@open.ac.uk

Universidad Politécnica de Madrid (UPM)

Campus de Montegancedo sn
28660 Boadilla del Monte
Spain
Contact person: Asunción Gómez Pérez
E-mail address: asun@fi.upm.es

University of Liverpool (UniLiv)

Chadwick Building, Peach Street
L697ZF Liverpool
United Kingdom
Contact person: Michael Wooldridge
E-mail address: M.J.Wooldridge@csc.liv.ac.uk

University of Sheffield (USFD)

Regent Court, 211 Portobello street
S14DP Sheffield
United Kingdom
Contact person: Hamish Cunningham
E-mail address: hamish@dc.shef.ac.uk

Vrije Universiteit Amsterdam (VUA)

De Boelelaan 1081a
1081HV. Amsterdam
The Netherlands
Contact person: Frank van Harmelen
E-mail address: Frank.van.Harmelen@cs.vu.nl

University of Karlsruhe (UKARL)

Institut für Angewandte Informatik und Formale
Beschreibungsverfahren - AIFB
Universität Karlsruhe
D-76128 Karlsruhe
Germany
Contact person: Rudi Studer
E-mail address: studer@aifb.uni-karlsruhe.de

University of Manchester (UoM)

Room 2.32. Kilburn Building, Department of Computer
Science, University of Manchester, Oxford Road
Manchester, M13 9PL
United Kingdom
Contact person: Carole Goble
E-mail address: carole@cs.man.ac.uk

University of Trento (UniTn)

Via Sommarive 14
38050 Trento
Italy
Contact person: Fausto Giunchiglia
E-mail address: fausto@dit.unitn.it

Vrije Universiteit Brussel (VUB)

Pleinlaan 2, Building G10
1050 Brussels
Belgium
Contact person: Robert Meersman
E-mail address: robert.meersman@vub.ac.be

Work package participants

The following partners have taken an active part in the work leading to the elaboration of this document, even if they might not have directly contributed to writing parts of this document:

Centre for Research and Technology Hellas
École Polytechnique Fédérale de Lausanne
France Telecom
Free University of Bozen-Bolzano
Freie Universität Berlin
Institut National de Recherche en Informatique et en Automatique
Learning Lab Lower Saxony
National University of Ireland Galway
The Open University
Universidad Politécnica de Madrid
University of Innsbruck
University of Karlsruhe
University of Liverpool
University of Manchester
University of Sheffield
University of Trento
Vrije Universiteit Amsterdam
Vrije Universiteit Brussel

Changes

Version	Date	Author	Changes
v0.1	23.08.04	Stefano Spaccapietra	creation
v0.2	18.11.04	Stefano Spaccapietra	First version of Part 1
v0.3	20.01.05	Stefano Spaccapietra	Part 1 completed
v0.4	08.03.05	Stefano Spaccapietra	Part 1 consolidated
v0.5	24.05.05	Stefano Spaccapietra	first draft part 2
v0.6	10.06.05	Stefano Spaccapietra, Andrei Tamilin	integration of contributions to part2
v0.7	15.06.05	Stefano Spaccapietra, Andrei Tamilin	different contributions compiled together, working draft for NoE meeting in Crete
v1.0	09.07.05	Stefano Spaccapietra, Holger Wache	version sent for review
v1.1	30.07.05	Stefano Spaccapietra, Holger Wache	final version

Executive Summary

Modularization is one of the techniques that bear good promises of effective help towards scalability in ontology design, use, and management. Currently, the issue of modularization in ontologies is very unresearched and very open. This deliverable has to be understood as exploring an area where much research is still preliminary stages, identifying many of the different open topics with the view of preparing building an overall framework in which all the work fits.

This deliverable continues the effort on modularization by Working Group 2.1 on scalability. The previous deliverable has outlined the main advantages expected from modularization and developed a review of state-of-art research in the field.

With this deliverable the working group sets the scene for a deeper understanding of what modularization may mean in the world of ontologies, and explores the relevant efforts and results achieved by KWeb partners, with the view that this exploration will eventually lead to some convergence of efforts.

The deliverable is organized into two main parts. The first one discusses the concepts and issues related to modularization. This was felt necessary, as it is easy to realize that a multiplicity of views over modularity exist in the different groups and in the literature, resulting in frequent confusion and misunderstanding. Part 1 is a contribution to organizing and clarifying the discourse on ontology modularization.

Part 2 reviews different approaches to ontology modularization. Its scope includes first the design phase, investigating how modules may be designed, characterized, assembled, and controlled. Description logics, graph algorithms, and conceptual modeling contribute to this part. Second, a number of proposals on reasoning in a modular ontology context are discussed. We are happy that the discussion can include a variety of approaches, from the distributed DL reasoning to CASE-based reasoning and query processing techniques.

The results from this work show that the partners are now in a perfect position to understand each other's work, and appreciate differences and complementarities. This is the cement that will promote mutual enrichment and enable each partner to continue developing its approach in coordination with the work by the other partners.

Contents

I	GENERAL FRAMEWORK	1
1	Introduction	2
2	Goals of Modularization	5
3	Module Definition and Description	8
4	Modularity Criteria	10
5	Properties of Modules and Modularization	12
6	Inter-Module References and Module Composition	14
7	Module Overlapping and Conflicts	16
8	Conclusion	18
II	MODULARIZATION APPROACHES	19
9	Overview of technical contributions	20
10	Partitioning	23
10.1	The Partitioning Method	24
10.2	Partitioning OWL Ontologies	25
10.2.1	Partitioning of the Class Hierarchy	25
10.2.2	Using Domain Relations	27
10.3	Tool Support for Automatic Partitioning	29
10.3.1	Graph Generation	30
10.3.2	Partition Generation and Improvement	31
10.3.3	Using Pajek as a Tool for Analyzing Ontologies	33
10.3.4	Automatic Comparison	34
10.4	Discussion	35
11	Modularization for Scalable Ontology Engineering	36

11.1	A Simple Example	36
11.2	Synthesis of Related Work	39
11.3	Our Approach	39
11.3.1	Modularity Criterion (Decomposition)	40
11.3.2	Module Composition	40
11.4	Formal Framework	42
11.4.1	Definition (Module)	42
11.4.2	Definition (Model, Module Satisfiability)	42
11.4.3	Definition (Composition Operator)	42
11.4.4	Definition (Modular Axiomatization)	45
11.5	Conclusion and Implementation	45
12	Engineering Robust Modules	47
12.1	Overview	47
12.2	Primitive Skeleton	48
12.3	Rationale	51
12.4	Discussion	52
12.5	Issues and Problems	55
12.6	Conclusion	56
13	A Contextualization Mechanism for Modularization	58
13.1	Stamping	60
13.2	Multiple Representations Modeling	61
13.3	Context-Varying Relationship Types	64
13.4	Context-Varying Is-a Links	65
13.5	Context-aware Querying	66
13.6	Conclusion	68
14	Distributed and Modular Ontology Reasoning	70
14.1	Distributed Description Logics	71
14.2	Inconsistency in DDL	75
14.3	Fixed-Point Semantics of Bridge Rules	77
14.4	Distributed Tableaux Algorithm for DDL	80
14.5	DRAGO Reasoning System	81
14.6	Preliminary Evaluation of Distributed Reasoning	85
14.7	Conclusions and Outlook	86
15	Reasoning on Dynamically Built Reasoning Space with Ontology Modules	89
15.1	Introduction	89
15.2	Ontology Space and Modules	90
15.3	Reasoning Space	91
15.3.1	Ontology Query Model	93
15.3.2	Finding Relevant Entities on the Ontology Space	93
15.3.3	Answering Queries over the Reasoning Space	94

15.3.4	Dealing with Global Interpretation	94
15.4	Applying the Reasoning Space Approach into a Use Case	95
15.5	Conclusion	97
16	Decentralized Case-Based Reasoning with an Application to Oncology	98
16.1	Introduction and Motivation: Adaptation Within Multiple Viewpoints in Oncology	98
16.2	Case-Based Reasoning with OWL	100
16.2.1	Principles of Case-Based Reasoning	100
16.2.2	Reformulations: an Approach for Representing Adaptation Knowl- edge	100
16.2.3	CBR within OWL ontologies	101
16.3	Decentralized Case-Based Reasoning with C-OWL	102
16.3.1	CBR with Contextualized Knowledge	102
16.3.2	Combining Viewpoints Thanks to Bridge Rules	103
16.4	Application to Breast Cancer Treatment	106
16.5	Discussion and Related Work	108
17	Conclusion	110

Part I

GENERAL FRAMEWORK

Chapter 1

Introduction

by STEFANO SPACCAPIETRA

KnowledgeWeb Deliverable 2.1.1 [WSC⁺04] has analyzed various techniques to achieve scalability in ontology management systems. This deliverable is meant to further explore the technique known as modularization. This issue was introduced in the previous deliverable using the following terms:

”In order to deal with the envisaged volumes of information, new technologies will be required. We will focus on knowledge process and ontology-based tool benchmarking. Related to knowledge process, we will explore new techniques for approximation (in order to reduce computational costs) and modularity (in order to reduce the amount of information that must be taken into account).”

As the name indicates, modularization has to do with modules. An intuitive understanding of the concept of module is some subset of a whole that makes sense (i.e., is not an arbitrary subset randomly built) and can somehow exist separated from the whole, although not necessarily supporting the same functionality as the whole. Chapter 3 hereinafter discusses the issue of finding a more proper definition of a module. Of course, the ”whole” this deliverable is interested in is an ontology, so our modules are ontological modules.

Even at this informal level, it is worth, to limit ambiguities, making a difference between two types of components of a whole: a module and a part. A module is a component that is expected to somehow support similar functionality as the ones supported by the whole. For example, an ontological module conveys knowledge in a form that supports reasoning. How this concept of similarity is exactly defined is an open issue. For example, a small ontology can be a module of a larger ontology. A part is a component that specializes in some of the functionality offered by the whole. For example, the T-box and A-box of an ontology are two different parts of the ontology. Though modules and components are not the same thing, they share a lot and this should be mentioned and explained.

The development of proper ontological modules should provide a mechanism for packaging coherent sets of concepts, relationships, axioms, and instances, and a means for reusing these sets in new environments, possibly heterogeneous with respect to the environment the modules were first built. To enable the reuse of such modules, a description of their functionality ("competence") is necessary. For example, the description could include syntactic items (e.g., information on language choices and commitments to paradigms and modeling styles) and semantics items (e.g., keywords characterizing the content of the module). Notice that for this description an ontology can be used, thus allowing to automatically detect which module can be used to answer a query. Description of modules is also addressed in Chapter 3 hereinafter.

Modularization can be perceived in three different ways. On the one hand, people think of modularization as the process that leads to decomposing a large ontology into smaller modules. The starting point is the whole ontology; the target is the modules. Proposals for a methodology to implement a decomposition strategy include those reported in the second part of this deliverable (Chapters 10 and 12). On the other hand, an equally viable perception is to assume that the semantic web is filled with ontology modules and that there is a requirement to assemble some of these modules to form a wider ontology. The module in this perception is something like a building block. The starting point is the set of useful modules; the target is the new ontology. This kind of modularization requires the specification of mechanisms to construct new ontologies from modules, e.g., inclusion operators, mapping rules and redefinition methods. Chapter 11 proposes such a composition mechanism. Actually, some of the modules used as building blocks may originate from the decomposition of an ontology, making the two approaches (composition, decomposition) coexisting within the same environment. As an ontology may be built up from other ontologies written in different representation languages, the characterization of modeling primitives in different languages may be necessary. A third alternative perception locates modularization at the design level. As suggested in Chapter 11, the hypothesis is that the ontology designer/builder knows about the target modules and while specifying ontology items (s)he also specifies to which modules they belong. In this approach, modularization is performed on the fly, as a by-product of design, and there is neither decomposition nor composition. The ultimate result is not necessarily the same as the one obtained by first designing the whole ontology and second splitting it into modules.

The three interpretations are discussed in this deliverable. However, coverage of the assembling approach is limited, as the approach is the focus of deliverables from WP 2.2.

According to this description, further work is needed to clarify the following issues:

1. What kind of modularization does make sense? The aim is to reduce the amount of information that must be taken into account. This entails that it must be possible to satisfy information requests by looking at only one (or a few) module. Adequacy between a given modularization and user/application requirements is a very open question which does not seem to have been addressed up to now.

2. A module is a packaging for a coherent set of concepts, relations, axioms, and instances. How can we define what "a coherent set" is? Chapters 10 and 12 offer at least a possible partial answer to this question. The former adopts a more mechanical definition, based on structural properties of the ontology, while the latter seeks for a more semantics-driven definition assisted by the knowledge of the ontology designer.
3. How is a module described? Potential users have to find out which module to use. The choice may be driven by the content of the module, the paradigm and formalism it uses to organize its content, the language it speaks. etc. The question is partially in the hands of research on service description languages. It has not been addressed in the works reported in this deliverable.
4. How can modules possibly be linked to each other, and what kind of inter-modules links would be desirable? How can mappings between modules be defined? How are they used? A concrete proposal is reported in detail in Chapter 14.
5. A particular use of a module is as a component contributing to building a new ontology. How does this composition operate? How does it take into account the mappings defined at point 4? Chapter 11 offers a possible answer.

Discussion of these questions forms the framework to understand the many facets of modularization. The following chapters in this first part of the deliverable provide the reader with such discussions. The second part of the deliverable is devoted to the analysis of some of the many techniques that contribute to solving the issues raised by a modular approach to ontologies.

Chapter 2

Goals of Modularization

by STEFANO SPACCAPIETRA

The understanding of what modularization exactly means, and what are advantages and disadvantages that can be expected from modularization, depends on the goals that are assigned to modularization. This chapter lists the possible goals we highlight.

Scalability This is an all-embracing goal, which sees modularization as a way to keep performance of DL reasoners at an acceptable level. The basic idea is that reasoners are known to perform well on small-scale ontologies, with performances degrading rapidly as the size of the ontology increases. So, if the amount of ontological knowledge to be analyzed for a given reasoning task can be kept small in all cases (or at least in a majority of cases), performance will be acceptable. Modules help in this, although there is no commitment that the size of a module is small enough to make reasoning realistically possible. Also, it remains to be demonstrated that, whenever a given reasoning task requires a network of modules to be searched, the overall time for coming up to a result is less than the time required for the same task executed against a single ontology formed by turning the modules in the network into a single ontology.

The scalability goal is most naturally associated with the decomposition approach, i.e. it materializes into the fact that an ontology gets split into smaller modules. However, scalability may also be a concern in a composition approach. In this case it materializes as the decision to keep existing modules as separate contributors to the desired knowledge set needed by some application, rather than integrating them to form the desirable ontology. This implies that some form of distributed reasoning should be available, as the tasks at hand will then operate on a network of ontology modules.

In the decomposition approach, scalability concerns may be split into two sub-topics, as follows.

- **Scalability for information retrieval**

For this goal, the driving criterion for modularization is to localize the search space

for information retrieval within the limits of a module. Implementing this decomposition criterion requires knowledge about the search requests that are expected. This type of knowledge can be extracted a posteriori from observing information requests over some period of time. Predicting this knowledge a priori would be more effective, but difficult to achieve (Chapter 11, for instance, assumes a priori knowledge is available with the ontology designer).

- **Scalability for evolution and maintenance**

For this goal, the driving criterion for modularization is to localize the impact of updating the ontology within the limits of a module. Implementing this decomposition criterion requires an understanding of how updates propagate within an ontology. It also requires knowledge on the steadiness of the information in the ontology. Steadiness here is meant to denote the likeliness of an evolution. A possible factor for steadiness is the confidence level attached to information in the ontology. How confidence levels, and more generically steadiness indicators, are acquired remains an open issue for research. Chapter 14 illustrates this approach.

Complexity management While scalability usually refers to performance in using the ontology, there is also an identical issue regarding the design of the ontology. The larger the ontology, the more difficult is controlling the accurateness of the design, especially if the designers are humans. It has been suggested that an easier to follow approach is to have designers designing ontology modules of a size designers can apprehend, and later compose these modules into the final ontology. This is one more illustration of the divide-and-conquer principle.

Understandability When encountering an ontology, the first issue at hand is to be able to understand its content. Of course, this is easier if the ontology is small. This is undoubtedly true if the user of the ontology is a human being, but also holds if the user is an intelligent agent navigating through the Web-services space. Size, however, is not the only criterion that influences understandability.

Personalization Ownership of information is known to be an important factor to be taken into account when organizing a cooperative system. This may also apply to ontologies, although most of them as seen as publicly available resources. Ownership in these cases provides the criterion for decomposing the ontology into smaller modules. Ownership information can also be attached to existing module to make them complying to a personalization environment. The technique discussed in Chapter 13 holds a possible response to the need for personalization.

Reuse Reuse is a well-know goal in software engineering. Reuse is most naturally seen as an essential motivation for the composition approach. However, it also applies to the

decomposition approach, where it would lead to a decomposition criterion based on the expected reusability of a module (e.g., how well can the module fill purposes of various applications?). Reusability emphasizes the need for rich mechanisms to describe modules, in a way that maximizes the chances for modules to be understood, selected and used by other services and applications.

Chapter 3

Module Definition and Description

by STEFANO SPACCAPIETRA

Although ontology management tools and reasoning services can operate on an ontology consisting of a single axiom, from a usefulness perspective a module cannot just be an arbitrary subset of an ontology. Indeed, while complying with the formal definition of an ontology, e.g. as a set of concepts, relations, axioms, and instances, an arbitrary subset does not comply with the goals assigned to ontologies. In particular, queries to such an arbitrarily modularized ontology would need examining all modules, one after the other, till the required information is found.

A module is therefore defined as a sub-ontology that “makes sense”. It may make sense from the application perspective, i.e. the module is capable of providing a reasonable answer to at least some of the queries it is intended to support. Alternatively, it may make sense from the system perspective, i.e. the modular organization is capable of improving the performance of at least some of the ontology management services. The vagueness of this definition reflects the subjective nature of the decision about what could be and what could not be regarded as a module. This vagueness, however, does not prevent the concept to be operational.

Also, it is possible to define some formal criteria to check that a given collection of ontology components makes sense or not. For example, it could be stated that a collection that includes instances not related to any concept (or concepts not related to any other concept) is not desirable. However, there is currently no agreement on criteria to separate good ontologies from not-so-good ontologies. This may be an item in a quality-of-service research agenda.

Defining a module as a sub-ontology translates the fact that an ontology is turned into a module when considering it in a wider framework where the targeted service is to be provided by a collection of modules. Conversely, a module can be considered as a self-standing ontology for purposes that do not require access to other modules in the collection.

Modules can be independently developed ontologies that are put together to form a

collection providing some new services. This is the composition approach to ontology modularization. Typically, such a modular architecture would include facilities for importing new ontologies into the collection, according to organizational rules characterizing the collection.

Modules can also be built by splitting an existing ontology. This is the decomposition approach. Splitting may be done manually, but is more likely to be done (semi-)automatically by the ontology management system, based on a decomposition criterion explicitly defined by the ontology administrators. Having the criterion explicitly stated also allows automatic maintenance of the collection of modules when modifications (insertions, deletions, updates) are introduced.

We say a module is "closed" if it does not contain any link to another module. A typical example of a closed module is an external ontology that has just been imported into a collection of modules.

We say a module is "open" if it contains links to other modules. Open modules require a choice of which interoperability techniques are to be implemented to support the collection of modules.

The same openness and closeness concept applies to collections of modules.

A collection of closed modules is likely to maintain some meta-ontology describing the scope of each module, so that the relevant modules can be identified when a query is addressed to the collection manager.

A collection of open modules may not need to maintain a meta-ontology external to the modules in the collection, but its organization may also include such a meta-ontology.

Chapter 4

Modularity Criteria

by STEFANO SPACCAPIETRA

In composition approaches, modules pre-exist the collection that forms the wider ontology they belong to. The question about what to put into a module does not arise, unless the strategy to integrate a module in the collection contains some rules to redistribute or reallocate the content of modules within the collection. For example, the strategy could instruct the ontology system to remove duplicate content and replace it by inter-module links, in an effort to reduce the individual or the cumulative size of the modules.

In the decomposition approaches, instead, finding a good decomposition criterion is a challenge. Relying on human "ontological commitment" is the simplest solution, but not a very satisfying one, as it makes the quality of service entirely dependent on the expertise of the ontology designers. While implementing this decomposition strategy, it would be recommended to also implement a complementary trust management component, so that reliability and efficiency of services provided by modules can be monitored, possibly leading to some preference ordering or other forms of reorganization among the modules in the collection.

In a human-based decomposition, rather than asking the ontology designer to position every ontology component (e.g., concept, relation, axiom, instance) into one or more modules, it is possible to ask the designer to identify group of components (e.g., groups of concepts) that have to be kept together, and then apply some algorithm that builds a module for each group with the selected components and the other components attached to the selected ones. Of course, the algorithm would have to use some criterion to determine how far it should go in looking for attached components (e.g., using something like a threshold for distance between the selected component and the other components).

Implementing an automatic or semi-automatic decomposition strategy on an application perspective requires knowledge about the application requirements. Such knowledge can be acquired, for example, by analyzing the queries that are addressed to the ontology and storing the paths within the ontology that are used to respond to queries. Frequency of paths and their overlapping can lead to determine the clustering rule that produces the

optimal decomposition. A number of techniques are available to do such data mining and clustering analyses. Path analysis is at the core of the decomposition approach proposed in Chapter 10.

A performance-based decomposition can be seen as a strategy that only considers system aspects, ignoring application requirements. This is not to say that application-based decompositions do not aim at improving performance. Examples of performance-based decompositions are graph decomposition algorithm. They aim at decomposing a graph into a collection of sub-graphs that shows the desired properties, and hold whatever the semantics of the nodes and edges in the graph is.

Chapter 5

Properties of Modules and Modularization

by STEFANO SPACCAPIETRA

When aiming at modular ontologies, questions about correctness are important open research issues. The problem can be split into correctness of modules (taken individually) and correctness of the collection of modules that forms the wider ontology.

The former has already been mentioned in previous chapters: What guarantees that a module is an ontology, and what guarantees that a module makes sense.

In composition approaches, correctness of the collection has to do with whether the composition has produced a semantically correct result. If, for example, the composition approach at hand aims at producing a fully integrated ontology, the correctness criterion can be that the resulting ontology contains a synthesis of all the components in the source modules (i.e., no information is lost in the process). It may also be requested that, in addition, the mappings between the resulting ontology and the input modules are defined. Alternatively, if, for example, the targeted result is the specification of the discovered links among modules, correctness may be defined as the fact that all relevant links are implemented and no implemented link is duplicated or inferable from the other links.

Correct solution of all kind of conflicts (syntactic and semantic heterogeneities, different granularities, etc) among the source modules is the key to a correct composition strategy.

In decomposition approaches, correctness of the collection again translates the fact that no information is lost in the process. Information preserving may be defined as the fact that the result of a query addressed to the collection is functionally (i.e., not from a performance viewpoint) the same as the result of the same query addressed to the original ontology. Another candidate correctness criterion is that after the decomposition the same inferences lead to the same results (which means information is preserved).

Information preserving can also be defined as the fact that, when recomposing the

original ontology from the modules (using some composition rules), what is obtained is exactly the original piece, nothing less, and nothing more. Depending on the decomposition rules used, it may be possible to guarantee that, if rules are obeyed, the decomposition they generate is information preserving.

If information loss cannot be avoided, an estimation of the information loss can be a very useful add-on for the query answering techniques (cf. Eduardo Mena's work on this topic [MI01]).

Chapter 6

Inter-Module References and Module Composition

by STEFANO SPACCAPIETRA

Even if we start with the idea that modules are independent sub-ontologies, it is unlikely that a module will always be able to produce a full answer to a query. There is a need for some facility for query answering based on multiple modules within the available collection of modules. This can be organized using a modular ontology management service that can identify the needed modules, query each module on the basis of its capabilities, and merging and synchronizing the different partial answers to form a global answer. In this case modules need not to be inter-related, i.e. modules may be closed, the links being maintained as metadata external to the modules and used by the query processing service. This metadata can be centralized in a single repository, or distributed and, for example, associated to each module as an interface definition which provides the necessary information on the content of the module and the way to retrieve its content (sort of encapsulation in the object-oriented sense).

In more cooperative (e.g., peer-to-peer) approaches, modules may be open, i.e. inter-related with one or several other modules, the links expressing paths to complementary information.

Links can be specified via assertional statements or via procedural statements. An assertional statement is a correspondence assertion that states a degree of commonality between components of one module and components of another module. Correspondences may be one-to-one as well as one-to-many and many-to-many. Correspondence assertions indicate that more about a component of a module can be found in the other module. Hence, whenever the query answering service finds that it needs more information than the one available in a given module, it could use these correspondence assertions to extend the scope of the search and continue its navigation in the linked modules.

Procedural specification of a link is frequently proposed as a view definition. The link is seen as a mechanism to extract some knowledge from another module, and what

is exactly extracted is defined as a query over the other module(s). Whenever the query answering mechanism needs information visible in the view that materializes the link, traditional query rewriting techniques are applied to the original query to generate the corresponding query over the other module.

The advantage of an assertional specification is that the method and the extent of the process extracting information from a module to enhance information available in another module can be dynamically determined during the query evaluation process. In other words, the strategy can be adapted to the actual query. The procedural specification of a link as a view is easier to operate, and may lead to better performance in query evaluation. Its disadvantage is that it is one and the same for all possible queries, so there is a dimension of personalization that is lost.

As links define potential or actual mappings among modules, an important question is to determine what kind of mappings can be dynamically evaluated. For example, assume a mapping specifies that a concept A in a module M_i corresponds to the union of concepts B and C in another module M_j , and that a relationship $r1(C, D)$ between C and D holds in module M_j . Then, to evaluate a query Q that references A in M_i , we might want to traverse the mappings between M_i and M_j to find out what does M_j have to say about B and C . It might be the case that $r1$ restricts C , which corresponds to a restriction of its associated concept A . A possible query answering strategy based on these mappings is discussed in Chapter 15.

What are the possible heterogeneities that can be supported in this context?

The answer to this second question depends on the mapping language in use. Some mappings might not allow for dynamically evaluation, requiring, eventually, the materialization of such mapping results, with the entire burden needed to keep those in synchrony.

Chapter 7

Module Overlapping and Conflicts

by STEFANO SPACCAPIETRA

Modules built by decomposition result from an initial unique perception of the world (the one portrayed by the initial ontology). Therefore, it is reasonable to assume that they will show no representation conflicts, and they if any overlapping exists it has been planned for and is under control of the system.

On the contrary, modules in a collection built by composition are expected to show all kind of heterogeneities, due to different perceptions of the real world. These heterogeneities range from the formalism on which the module is built (e.g., one module could be built on RDF, another one on OWL), to the semantic content of the module (e.g., one module holds road network ontology while another one holds public transport system ontology). Despite the multiplicity of the sources the modules come from, the fact that they are composed to build a wider ontology most likely entails that there is some level of commonality among the modules. In particular, one would expect that modules show some semantic overlapping (i.e., some of them include description of the same piece of reality), with both duplicated and complementary elements within these intersections. The literature on composition (usually referring to the issue as data, information and ontology integration) has coined the term “conflict” to denote situations where representations of the same fact in different repositories are not identical, despite the fact that having different perceptions of the same thing is not per se a conflict in the informal use of the term. Therefore, conflict discovery and resolution are the core issues in any composition effort.

Taxonomies of conflicts are available in the literature (e.g. [SK93]). They usually distinguish such broad categories as terminological conflicts, description conflicts, structural conflicts, and semantic conflicts.

Some conflicts simply stem from terminological choices. An identifying property for a car concept, for example, is either named “CarId”, or “car_id”, or “Chassis#”, or “Chassis”. Terminological tools will easily identify the first two and the last two as being equivalent terms. But finding the equivalence between “CarId” and “Chassis#” requires knowledge that both fulfill the same identification role and both have the same value

domain.

Additional conflicts come from the fact that modules associate different descriptions, e.g. different properties, to the same concept. Car has properties <Chassis#, category> in one module, and has properties <CarId, Branch, Model, Make, Category, Year, Mileage, LastServiced> in another module.

Structural conflicts may be illustrated considering customer information. Assume a module holds a Customer concept which includes a "Rating" property (with value domain: "Preferred", "Blacklisted", etc) to discriminate various categories of customers. Another module can represent the same reality by having two sub-concepts (Blacklisted, FrequentTraveller) to the Customer concept. This difference between the two representations is mainly due to different levels of interest for the same fact in the two modules. Notice that a third module also holding the Customer concept may define two other sub-concepts of Customer to be Person and Company, the latter with sub-concepts Private-Corporation and Government.

Semantic conflicts stem from different classifications scheme, not involving subsumption links. One module can hold two concepts for bookings: one for current bookings (those where a specific car has been assigned), another for non-current bookings (where only a car category is specified). Another module has all bookings in a unique Rental-Booking concept. Current bookings are found by restricting Rental-Booking instances to those that are linked to a car instance by the Allocate-to role.

These differences give an idea of the complexity inherent to the composition process, which will have to sort out differences to build a consistent ontology. They also point at the benefit expected from the new ontology. The latter enables users to query, for example, the model of a car, while the car belongs to one module and its model to another module.

We do not further discuss this important topic, as it is central to the work in KWeb Workpackage 2.2, and refer the interested reader to deliverables from that Workpackage.

Chapter 8

Conclusion

by STEFANO SPACCAPIETRA

Managing large ontologies is a challenge for ontology designers, reasoners and users. A known approach to deal with large problems is modularization, in which a whole is organized into smaller parts that can be independently manipulated and still collaborate for the whole picture. Applying the modularization principle to ontologies should consider particular characteristics of the problem, such as, for example: the semantics of an ontology module, the criteria to achieve modularization, the necessary descriptive information of an ontology module, and the techniques to manage heterogeneity between ontology modules.

The challenge is high, and has already prompted attention by different research groups. However, the domain is not yet mature, and it can be easily acknowledged that discussions on ontology modularization rapidly get blurred by the fact that the concept can be understood in rather different ways. We have developed in the previous sections an analysis of the various facets and perceptions of ontology modularization. We have focused on identifying and showing alternative approaches, with their underlying assumptions as well as with their specific goals.

While it is too early to come up with consensus on an overall framework in which all the work fits, this first part of the deliverable was meant to make a first step hopefully into the right direction.

Looking for an overall framework does not mean that we think in terms of a future coherent single approach that would clearly outperform any other one. The need for modularization emerges from different contexts, characterized by different requirements. A multiplicity of solutions is required to cover all potential useful contexts.

Part II

MODULARIZATION APPROACHES

Chapter 9

Overview of technical contributions

by STEFANO SPACCAPIETRA

The many questions that Part I of this deliverable has identified and introduced are open research issues that have been or are being addressed by a number of research groups within and beyond the KnowledgeWeb community. This second part of this deliverable reports on ongoing efforts that in one way or another contribute to the goal of making the idea of modular ontologies operational. Each chapter presents the work of a KnowledgeWeb research group. At last, some pointers to related work by other groups are given. The conclusion summarizes the main achievements and points at directions for further efforts and recommendations for continuation of work within KnowledgeWeb.

The contributions hereinafter can be grouped into two sets, respectively focusing on:

- How modules may be created, assembled, and organized (Chapters 10 to 14); and
- How reasoning may be performed within a modular organization of ontological knowledge (Chapters 15 and 16).

The next two chapters illustrate the alternatives outlined in Part 1 Chapter 1, i.e. the fact that interest in modularization may focus on either how an ontology can be split into modules (by semi-automatic decomposition or by human design choice) or on how, given a set of existing modules, they may be composed to form a larger ontology:

- How to semi-automatically decompose an ontology into modules (Chapter 10, contributed by M. Menken, H. Stuckenschmidt, and H. Wache).

The proposed approach relies on the rewriting of the ontology as a weighted graph, on which a graph-partitioning algorithm is applied. As the most appropriate partitioning policy is difficult to determine, the algorithm is run repeatedly with changing input parameters. Results are then compared to infer which partitioning are the most likely good ones.

- How to compose modules, defined by an ontology designer using the ORM model, to form a new ontology (Chapter 11, contributed by M. Jarrar).

In this approach modules are seen as relatively small, shared ontologies covering some generic and limited domain and based on a common terminology, to be used as building blocks for elaborating larger ontologies for wider domains. A composition operator is defined for merging of modules. Consistency of the composed result is checked.

The following chapter also follows the decomposition approach, but focuses on recommending that an ontology be restructured before decomposing it. The purpose of the restructuring is to conform the ontology to a defined pattern that is believed to lead to a decomposition that is semantically meaningful to the applications using the ontology:

- Which criteria can be used to characterize a good module (Chapter 12, contributed by A.Rector and J.Pan).

The proposed approach defines a number of structural rules that are claimed to provide a sound basis for modularization. The approach parallels database normalization approaches in that the original ontology is restructured to make it compliant with the rules that have been defined.

The last contribution in the first set proposes a mechanism to identify modules within an ontology. The goal is to preserve the original ontology as a whole, while allowing one or more decompositions into overlapping modules that can be used separately (for queries confined to a given module), or together (for queries spanning over different modules):

- How ontology modules can coexist within a single ontology with controlled sharing of ontology elements (Chapter 13, contributed by C.Parent and S.Spaccapietra).

The focus in this approach is to provide rules for describing how information in an ontology relates to one or more modules. Each element of an ontology can be defined as relating to multiple modules, sharing the element. The sharing can be from the meta-level (T-box) to the value level (A-box). Alternatively, elements may be defined as specific of a single module. The repository (T-box+A-box) can then contain both the original ontology and its modules. The same mechanism can be used to define context-dependent ontologies.

Contributions in the second set offer alternative approaches to reasoning with multiple ontologies, which includes reasoning on modular ontologies. The common goal is to enable performing a reasoning task without having to a priori centralize all available knowledge into a single ontology:

- Reasoning with multiple ontologies connected via directional links (Chapter 14, contributed by L.Serafini and A.Tamilin).

The proposed reasoning formalism is based on directional mappings between ontologies. Each mapping defines the possibility for an ontology to get related knowledge from other specified ontologies. A formal framework is proposed to perform distributed reasoning using these inter-ontology mappings.

- Query answering over a modular ontologies space (Chapter 15, contributed by F.Porto).

This proposal looks at collaborative query answering in an ontology space over a peer-to-peer network.

- Case-Based Reasoning over multiple ontologies and across multiple contexts (Chapter 16, contributed by M.d'Aquin, J.Lieber and A.Napoli).

Like in the proposal by Serafini and Tamilin, this last contribution focuses on enabling adaptive reasoning in a multi-ontology environment. The chosen formalism is case-based reasoning. Semantic relations between contexts support reuse of knowledge across contexts.

Chapter 10

Partitioning

by MAARTEN MENKEN, HEINER STUCKENSCHMIDT, HOLGER WACHE

The work reported in this chapter aims at the management of large ontologies. Today large ontologies which we find for example in medicine or biology are available whose size and domain coverage make them very difficult to understand and manage. Our aim is to develop methods that automatically partition large ontologies into smaller modules that contain semantically related concepts. This aim is very challenging as the semantic relatedness of concepts is much more difficult to determine than coherence in a software system where we can look for function calls across modules.

There is some previous work on partitioning knowledge models. Amir and McIlraith [AM05] describe a partitioning method for logical theory that optimizes a distributed decision theory. Such a partitioning aimed at efficient reasoning, however does not necessarily correspond to a partitioning based on semantic relatedness. In fact these two goals seem to be orthogonal with respect to the optimal partitioning.¹ Mehrotra and others [MW95] discuss the partitioning of knowledge bases (mostly rule bases) with respect to different viewpoints. This work is more related to our problem. Currently, we do not address the problem of creating a partitioning according to a particular viewpoint. We rather try to find a generally applicable partitioning that helps to present the content to the user in a structured way.

This chapter continues previous work reported in deliverable Del22 "Ontology Refinement — Towards Structure-Based Partitioning of Large Ontologies" of the EU funded project *WonderWeb — Ontology Infrastructure for the Semantic Web* [SK04a]. In that deliverable we described a partitioning method that uses techniques from network analysis to partition simple class hierarchies based on their structure. But in domains like medicine, however, existing ontologies consist of far more than a simple hierarchy and make use of the expressive power of the web ontology language OWL. The problem we address in this chapter is how to adapt our method for partitioning simple class hierarchies

¹McIlraith 2004, personal communication

to more expressive ontologies paying special attention to ontologies encoded in OWL. In Section 10.1, we briefly recall the basic steps of our partitioning method and refer to the corresponding definitions in the original publication. In Section 10.2, we describe an extension of the method to OWL ontologies. In particular, we discuss different options for including the definitions of concepts into account during partitioning. In order to support this extended method we are developing a software tool that takes RDF Schema and OWL ontologies as input and makes proposals for a partitioning of the corresponding ontology. This tool is described in Section 10.3. We conclude with a discussion of the method and its potential role on the semantic web.

10.1 The Partitioning Method

In [SK04b] and [SK04a] we presented a method for automatically partitioning lightweight ontologies. In particular, the method was aimed at models that only consists of a concept hierarchy. We showed that using simple heuristics, we can create meaningful partitions of class hierarchies for the purpose of supporting browsing and visualization of large hierarchies. We briefly recapitulate the different steps of our method as the following discussions will be based on this information.

Step 1: Create Dependency Graph: In the first step a dependency graph is extracted from an ontology source file. The idea is that elements of the ontology (concepts, relations, instances) are represented by nodes in the graph. Links are introduced between nodes if the corresponding elements are related in the ontology (cf. [SK04a], page 4).

Step 2: Determine strength of Dependencies: In the second step the strength of the dependencies between the concepts has to be determined. This actually consists of two parts: First of all, we can use algorithms from network analysis to compute degrees of relatedness between concepts based on the structure of the graph. Second, we can use weights to determine the importance of different types of dependencies, e.g. subclass relations have a higher impact than domain relations. For our experiments (cf. [SK04a], page 4/5) we use the structure of the dependency graph to determine the weights of dependencies. In particular we use results from social network theory by computing the proportional strength network for the dependency graph. The proportional strength p_{ij} of a connection between a node c_i and c_j describes the importance of a link from one node to the other based on the number of connections a node has (a_{ij} is the weight preassigned to the link between c_i and c_j) [Bur92]:

$$p_{ij} = \frac{a_{ij} + a_{ji}}{\sum_k a_{ik} + a_{ki}}$$

Step 3: Determine Modules: The proportional strength network provides us with a foundation for detecting sets of strongly related concepts. This is done using a graph algorithm that detects minimal cuts in the network and uses them to split the overall graph in sets of nodes that are less strongly connected to nodes outside the set than to nodes inside (cf. [SK04a], page 5/6). For this purpose, we make use of the 'island' algorithm: A set of vertices $I \subseteq C$ is a line island in network if and only if it induces a connected subgraph and the lines inside the island are stronger related among them than with the neighboring vertices. In particular there is a spanning tree T over nodes in I such that [Bat03]

$$\max_{(u,v) \in V, v \notin T} w(u,v) < \min_{(u,v) \in T} w(u,v)$$

Step 4/5: Improving the Partitioning: In the last steps the created partitioning is optimized. In these steps nodes leftover nodes from the previous steps are assigned to the module they have the strongest connection to. Further, we merge smaller modules into larger ones to get a less scattered partitioning. Candidates for this merging process are determined using a measure of coherence ([SK04a], page 7, 9 and 10).

The strength of this method lies in its simplicity, generality and scalability. A valid question is now, if we can apply the same method to partition more complex (i.e. OWL-based) ontologies that we also find on the semantic web.

10.2 Partitioning OWL Ontologies

The application of this method to OWL ontologies raises further questions. In previous experiments we have only considered class hierarchies with no further definitions of classes or relations between them. In OWL ontologies, these additional definitions play an essential role, in particular, because they can be used to infer implicit subclass relations and therefore should be taken into account when determining modules. This can best be done by modifying the first step of the method, because all following steps work on the dependency graph created here. The problem of applying the partitioning method to OWL ontologies therefore reduces to the problem of adequately representing the dependencies implied by class and property definitions in a dependency graph. In the following, we discuss and compare different ways in which such a graph can be constructed for OWL ontologies using an example ontology.

10.2.1 Partitioning of the Class Hierarchy

The simplest way of creating the dependency graph for an OWL ontology is to do it in the same way we did for simple class hierarchy. In this case, we would ignore most of the

definitions of classes and just look at explicitly contained subclass statements that connect classes. This simple approach proved to work quite well for simple class hierarchies. When looking at the resulting dependency graph for our example ontology (Figure 10.1), we immediately see that this approach will often fail for OWL ontologies. The problem is that the nature of OWL allows us to build class hierarchies without explicitly using subclass relations. As a consequence, it will often happen, that a significant number of concept will only be linked with the TOP concept. When partitioning the ontology, all of these concepts will end up in the same partition. In the example ontology, we will only be able to distinguish between persons and animals, but the algorithm will not be able to create more fine grained partitions corresponding to different kinds of persons.

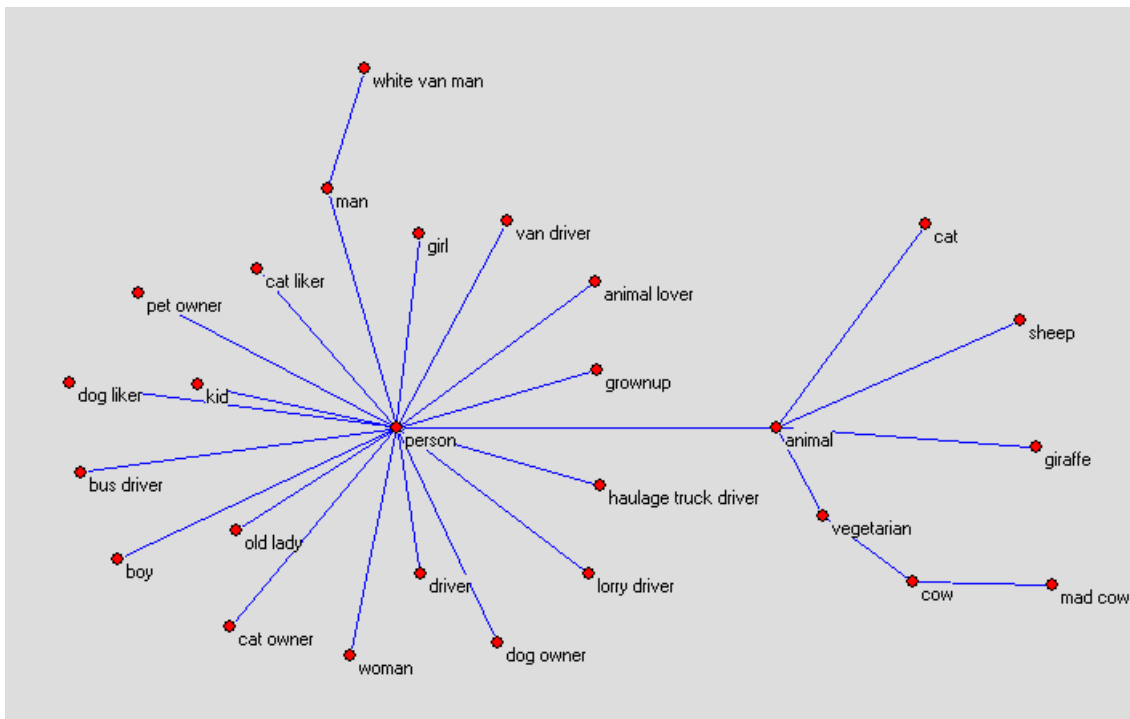


Figure 10.1: Dependency Graph of the Person-related Part of the Ontology

A straightforward solution to this problem is to not only consider explicit subclass relations, but to also include implied relations into the dependency graph. We can do this by simply computing the implied class hierarchy using an OWL reasoner. The derived hierarchy can be stored in the model. After doing this, we can use the same method for creating the dependency graph as we did before. The result of using the computed subclass hierarchy as a dependency graph for our example ontology is shown in Figure 10.2. We see that the different person-related concepts are now organized into subtrees with a coherent topic. We see for instance that different kinds of children and grown-ups are placed in different subtrees. Further, we see a distinction between concepts that represent

different roles of people (e.g. driver). This information can be used by the island algorithm to compute a more fine-grained partitions.

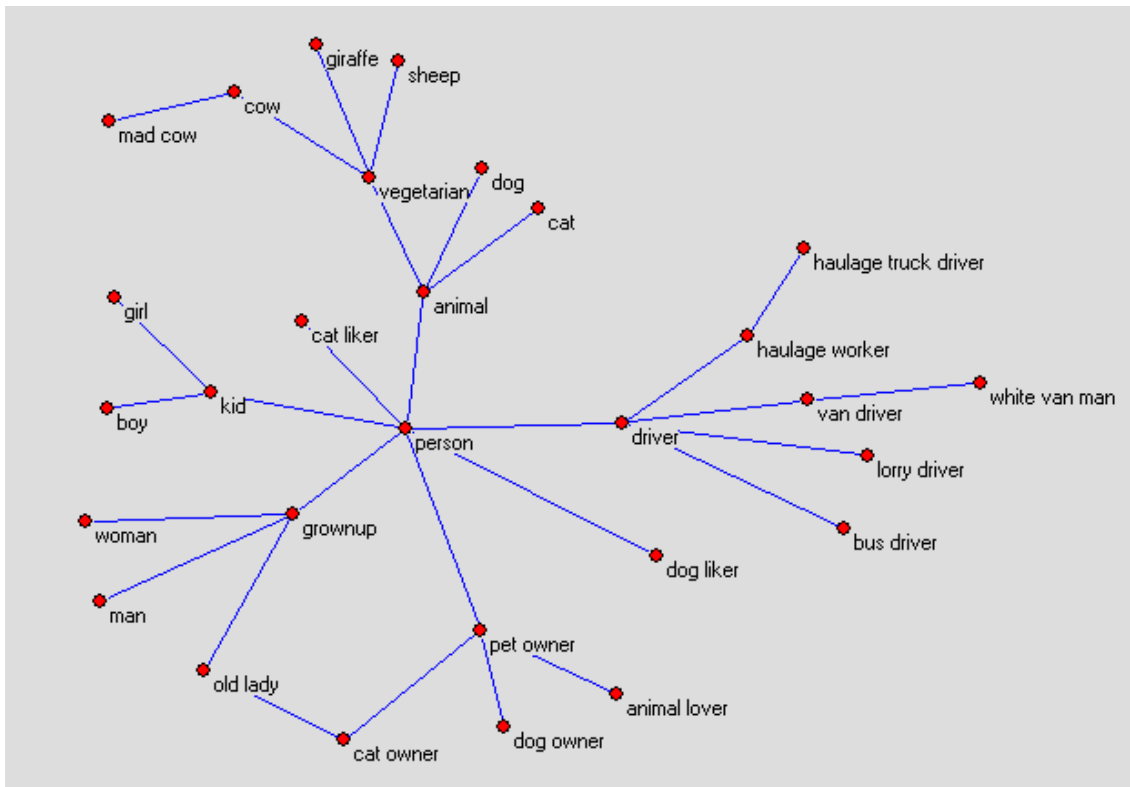


Figure 10.2: Dependency Graph after adding implied subclass relations

It might be argued that there is a conceptual difference between explicit and implicit subclass relations that need to be taken into account. We could consider to assign different levels of importance to explicit and implied relations. The level of importance can be encoded by weighted edges between concepts. These weights are input to the second step of the partitioning method and influence the relative strength between two nodes that we use as a measure for the partitioning. We will come back to the use of weights for different kinds or dependencies later.

10.2.2 Using Domain Relations

The idea of the previously discussed approaches for constructing the dependency graph relied on the impact that additional definitions have on the class hierarchy without directly encoding any of these definitions in the dependency graph. The question is now whether we can achieve even better results by also directly considering concept definitions. We

have to be careful, however, because our partitioning algorithm tends to not performing well on highly connected graphs. Therefore, if we want to include parts of the definitions into the dependency graph, we should restrict ourselves to the most significant parts of the definitions. An obvious choice is to include information about domain relations as they are an important part of an OWL ontology. Including relations in the dependency graph also has the advantage that they are assigned to partitions as well.

An obvious idea is to look at the domain and range of each relation and to declare the corresponding concepts to be dependent by virtue of being connected by a domain relation. This would mean treating domain relations in the same way as we treat subclass relations. Looking at this in more details, however, reveals that this way of using domain relations often produces undesired results. In fact, domain relations are often used to connect objects of a very different nature. In existing ontologies, the range of domain relations are often datatypes and even even object properties often link concepts that we would put into different partitions. In medical ontologies like the DICE ontology that we analyzed, an ontology would talk about things like Drugs diseases and body parts, each forming a coherent submodel. Domain relations that would typically be found are relations like 'treats' that connect drugs with diseases or 'located-in' connecting diseases and body parts. Adding these relations to the dependency graph would create connections between parts of the ontology that we want to keep separate.

Despite these problems, it turns out that we can use information about relations to determine additional dependency relations in a fruitful way. Reconsidering the medical example mentioned above, we conclude that concepts of the same nature are not characterized by being connected through a domain relation but by the use of a certain relation: Different kinds of Drugs will have the 'treats' relation in their definition, diseases will use the 'located-in' relation and body parts will often be defined using the part of relation. This observation is closely related to the use of implicit subclass relations as these relations are often computed by comparing restrictions on the same property. Using this shared use of properties as a criterion for dependency provides us with more information about related concepts but still abstracts from the details of the definitions often leading to a dependency graph with clear clusters. Figure 10.3 shows the dependency graph of our example ontology. It is based on the computed hierarchy and the shared use of relations. We can see that similar concepts are indeed grouped around certain relations (animals around 'eats', persons around 'sex' and 'age' pet-owners around 'has-pet' and drivers around 'drive').

Mixing different sources of dependency, in this case the concept hierarchy and the use of relations, brings us back to the problem of using weights for different types of dependencies. While we can justify the different sources of dependencies by observed modeling styles and language properties, there is no knowledge about the right choice of weights. Currently, we have to determine weights through systematic experiments apply-

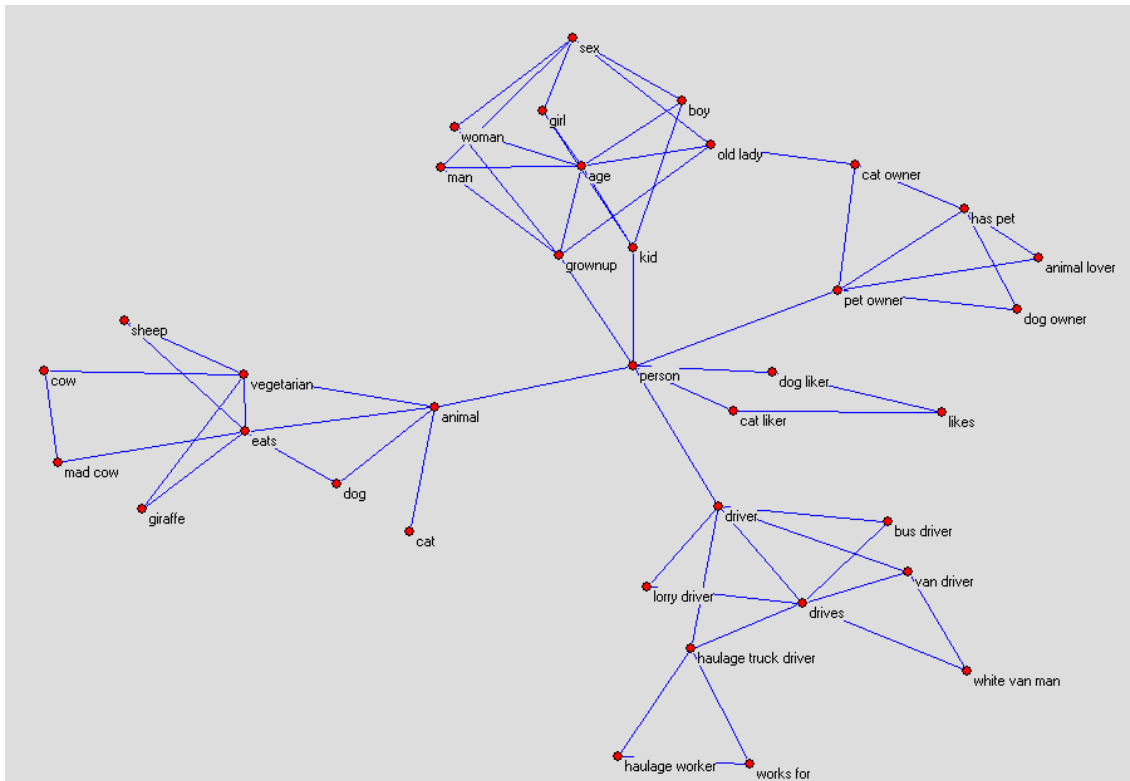


Figure 10.3: Dependency Graph after adding shared use of relations as a dependency criterion.

ing the method with different weights to the same ontology. In this way we can determine the best setting for a given problem. It is too early, however, to draw general conclusions about the importance of different kinds of dependencies and the corresponding weights to be used.

10.3 Tool Support for Automatic Partitioning

From the discussion about applying our partitioning method to OWL ontologies, we can derive a number of requirements for functionality that has to be provided by a partitioning tool. Besides the actual partitioning of the dependency graph these requirements are mostly concerned with the modification of input parameters and the evaluation of the resulting partitioning. A system for ontology partitioning has to provide options for different ways of generating the dependency graph based on the nature of the ontology to be partitioned. These options should at least include the options discussed above as well as the possibility to set weights for different kinds of dependencies. As we will have to run many experiments and compare their results in order to find the right weights, the system

should provide immediate graphical feedback in terms of the resulting partition for human inspection. For larger ontologies, this manual inspection will not be sufficient. Therefore, an automatic comparison function is needed that can compare the results of experiments to a golden standard and to each other.

Based on these requirements we have implemented a tool for supporting partitioning of OWL ontologies. It performs all the steps that were described before: creating the dependency graph, determining the modules, and improving the partitioning by assigning leftover nodes to the already found modules. The tool is a Java application that performs the partitioning interactively through a graphical user interface. It is freely downloadable from <http://swserver.cs.vu.nl/partitioning/> and licensed under the GNU General Public License.

10.3.1 Graph Generation

Our tool uses Sesame, a system for storing and querying data in RDF and RDFS [BKvH02]. The ontology is loaded into a local Sesame repository, after which it can be easily queried via an API. Because Sesame does not have native OWL support, some extra programming had to be done to deal with ontologies in this format. This includes explicitly querying for resources of type `owl:Class` while retrieving all classes (Sesame only returns resources of type `rdfs:Class`) and following blank nodes for determining the definition relations (see below). Further, irrelevant resources have to be filtered out. This is done on the basis of user-defined namespaces that are to be ignored. Resources that occur in those namespaces do not show up in the resulting network. In most cases the classes and properties defined in RDFS and OWL can be ignored because they do not describe parts of the domain but constructs for talking about them and are therefore on a different level of abstraction. By entering the corresponding namespaces in the text area also extensions of RDF or elements of other RDF-based languages can be excluded if they do not contribute to the actual domain model.

Before converting an ontology, the user has to decide what relations to include in the network, and if those relations are to be represented by edges (undirected) or arcs (directed). The tool allows five types of relations to be included: subclass, property, definition, substring, and string distance relations. The user also has to decide about the strength of each type. At the moment, only subclass relations that are explicitly stated in the ontology are included in the network. If the classified hierarchy is to be used for the partitioning, existing OWL reasoners can be used to compute implicit subsumption relation beforehand and add them as explicit statements into the repository. This can for example be done by using the BOR reasoner [SJ02] that is integrated with the Sesame system. When property relations are to be included, for each domain/range restriction a relation is created. Definition relations are established between a concept and its property

(or not only properties but also other resources). These can be used to make concepts dependent on some shared property. The remaining two relations, substring and string distance, look at the concept names (or labels if specified). They create a relation if one concept name is contained in another or if the string distance between two concept names is below a certain threshold.

Finding the definition relations was a bit cumbersome. Querying using the Sesame API involves posing queries of the form $\langle Subject, Predicate, Object \rangle$ and getting back a subset of triples from the ontology that match the query. The complicating factor is that the graph model representing the definition contains a number of nested blank nodes. See for an example Figure 10.4. Here, the definition relation should be created between the concept *dog* and the property *eats*. To find those relations, all triples in the ontology are retrieved and for each of those triples the blank nodes are traversed until an `owl:onProperty` statement is found.

```
<owl:Class rdf:about="http://.../mad_cows#dog">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="http://.../mad_cows#eats"/>
      <owl:someValuesFrom>
        <owl:Class rdf:about="http://.../mad_cows#bone"/>
      </owl:someValuesFrom>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Figure 10.4: Part of the example ontology showing relevant information for a definition relation. (*Adapted for readability.*)

Figure 10.5 shows a screen shot of the tool in which an OWL ontology is converted to a dependency network. The screen is divided into three parts: the upper part gives a short help text about the currently selected tab, the middle part is for specifying the required arguments (in this case the input ontology and the output network) and the bottom part is for various optional parameters that influence the conversion. The tool converts an ontology written in RDFS or OWL to a dependency graph, written in Pajek format.

10.3.2 Partition Generation and Improvement

Besides converting ontologies, another function of the partitioning tool is the creation of the partitions. Based on a dependency network, it creates one or more clusters. The maximum number of concepts per cluster can also be specified. The actual calculation of the

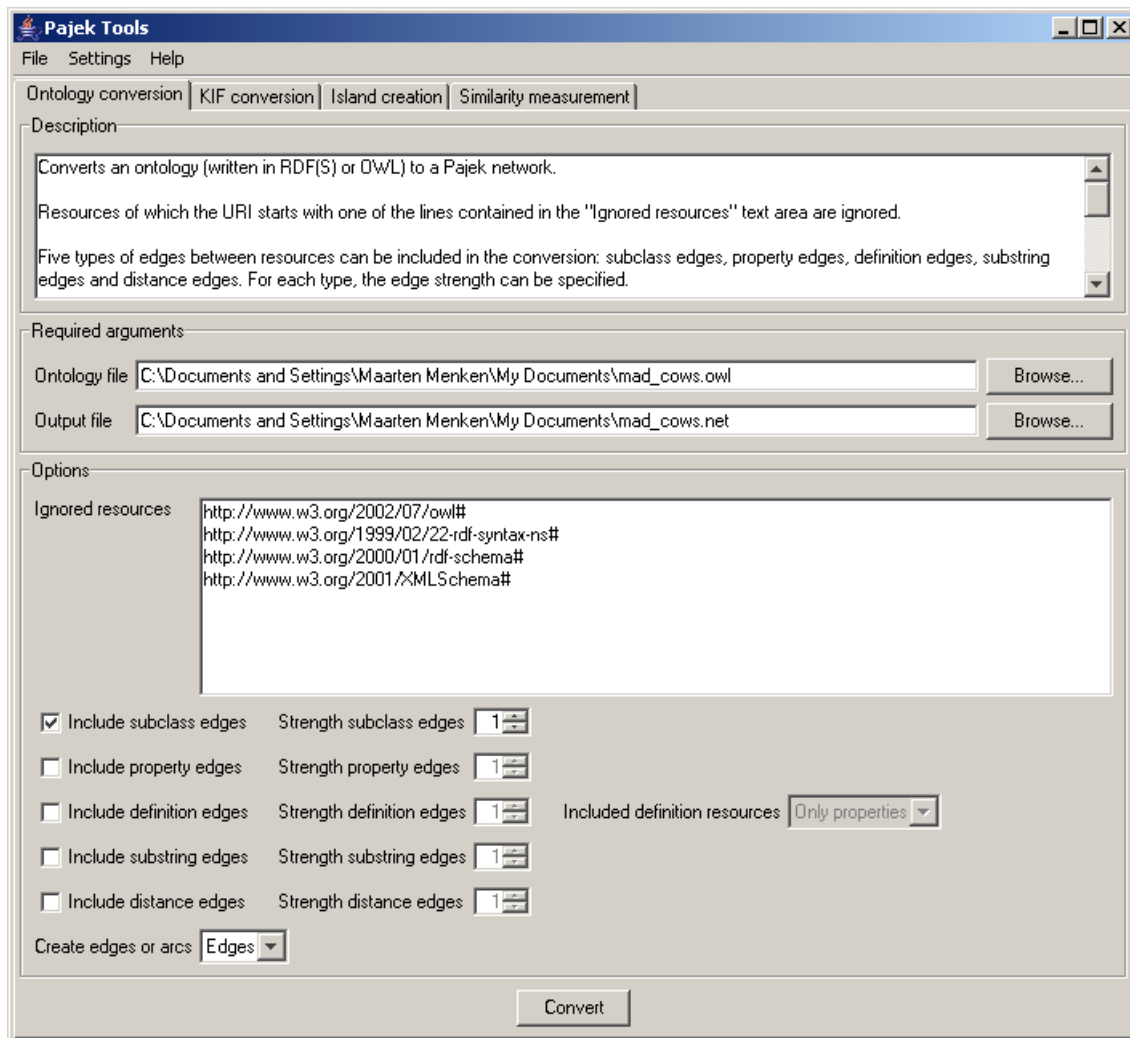


Figure 10.5: Screen shot of the partitioning tool with the ontology conversion tab active.

islands is done by an external Windows program written by Matjaz Zaversnik². Therefore this functionality is only available on Windows.

After this partitioning, in some cases there will be some leftover nodes which are not assigned to any cluster. The tool will automatically assign these nodes to the cluster to which they have the strongest connection. How this process works can best be explained by an example. Figure 10.6 shows an example network. It contains two modules (M1 and M2) and one leftover node (c8). c8 is connected to module M1 by one edge with strength 0.3 and to module M2 by two arcs with strengths 0.2 and 0.3. To determine the strength of a connection between a leftover node and a module, the strengths of all edges and arcs that connect the two are summed. Because edges are undirected and work in this respect

²<http://vlado.fmf.uni-lj.si/pub/networks/>

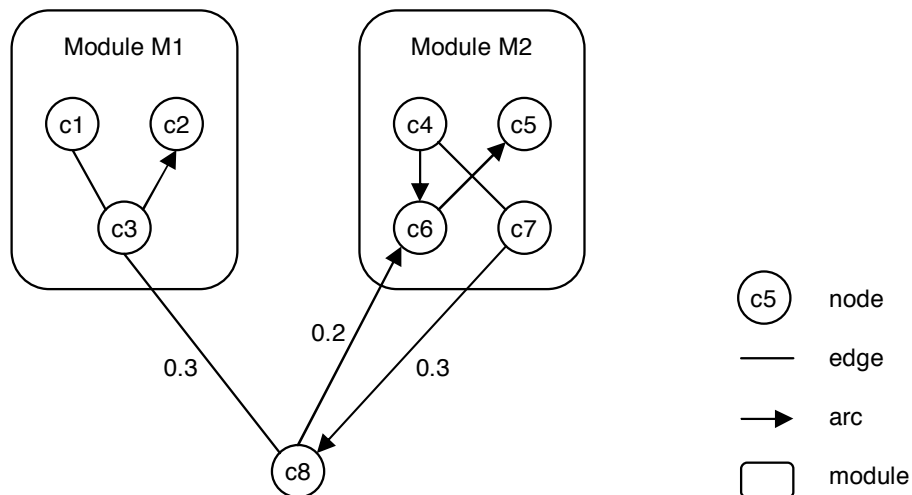


Figure 10.6: Example network for the assignment of leftover nodes to modules.

Another option for improving the partitioning is to merge small related modules into larger ones. The current implementation does not support this as there is still a need to explore this functionality on a theoretical level.

10.3.3 Using Pajek as a Tool for Analyzing Ontologies

The result of the conversion as well as the result of the partitioning process is represented in a format that can be processed by the Pajek network analysis tool [BM03] (available at <http://vlado.fmf.uni-lj.si/pub/networks/pajek/>). In particular, these results are:

- the dependency graph of the ontology
- graphs representing each individual module
- a graph containing the partitioning in terms of different node labels

These graphs can be loaded into Pajek for inspection and further processing. Figure 10.7 shows a screen shot of Pajek. A partition is shown in a graph, each module in a different color. In particular, the different graph layout algorithms of Pajek help to inspect the result of the process and spot potential problems. Pajek also provides functionality for managing and analyzing partitions. It often happens for example, that the dependency graph of the ontology contains different components. These components can easily be

detected and extracted by Pajek. The most interesting functionality, however is the possibility to compute different network theoretic measures like the internal coherence of a module that can be used to assess a created partitioning.

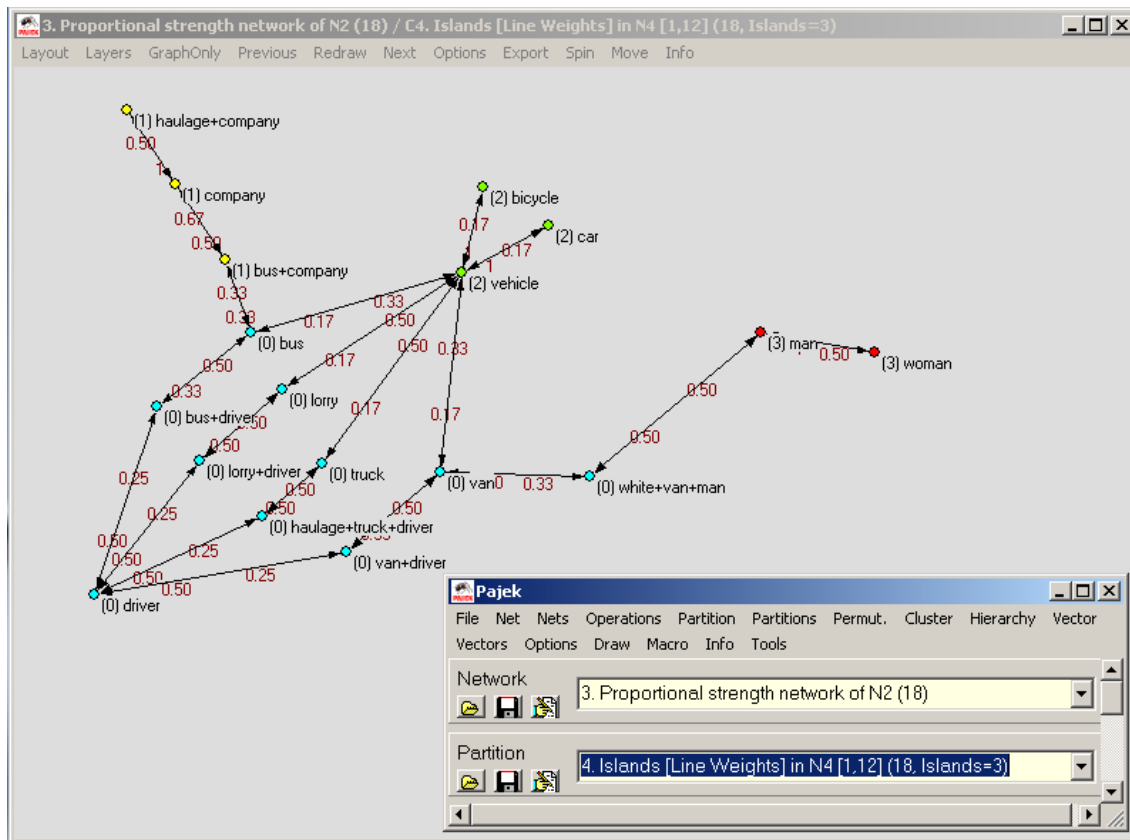


Figure 10.7: Screen shot of Pajek displaying four partitions.

10.3.4 Automatic Comparison

In the case of very large models the evaluation of the partitioning result can often not be done manually. For this purpose, our tool implements limited functionality for comparing two partitions of the same graph that can be used to compare the output of a partitioning process to a given partitioning. The measures implemented are adaptations of the classical precision and recall measures as well as a method called EdgeSim [MM01]. The first two measures are based on the numbers of intrapairs, which are pairs of concepts that are in the same cluster [AFL99]. Precision is defined as the percentage of intrapairs in the first cluster that are also intrapairs in the second cluster. Recall is defined as the percentage of intrapairs in the second cluster that are also intrapairs in the first. The EdgeSim measure considers both the vertices and the edges and is not sensitive to the size and number of clusters (as are precision and recall). Both intraedges (i.e., edges that connect

nodes within the same cluster) or interedges (i.e., edges that connect vertices in different clusters) are taken into account to calculate the edge similarity between two partitionings. The three measures give an indication of how well the partitioning was performed and therefore what relations and strengths give best results.

Currently we finish our implementation and will perform the first experiments on the several large ontologies.

10.4 Discussion

We presented a method and a tool for partitioning OWL ontologies based on different criteria for semantical relatedness of concepts. Using our tool, these criteria can be identified weighted and used to generate a dependency graph that serve as input for the actual partitioning method. This process of creating the dependency graph is the critical step in the method, because the success of the partitioning heavily depends on an appropriate choice of the parameters. In particular, we have to decide which criteria (subclass relations, shared used use of properties, etc.) to include and their relative importance in terms of weights. These choices are not trivial and will probably not be the same for any model. We believe that the only way of getting a better idea of the right choices is to carry out experiments on real ontologies. The tool described in this chapter cannot free us from this task, but it can ease the task by automating large parts of it. We are currently using the tool for carrying out experiments on a large medical ontology called DICE. We will test different strategies for creating the dependency graph and evaluate the result based on feedback from medical experts. At the moment it is unrealistic to assume that domain experts will be able to use the tool directly, because the choice of the parameters requires some knowledge about the partitioning method. Improving the tool far enough to enable other people to use it off the shelf nevertheless is the ultimate goal of this work.

Chapter 11

Modularization for Scalable Ontology Engineering¹

by MUSTAFA JARRAR

The main idea of the modularization in this section is to develop an ontology as a set of small modules and later (i.e. at the deployment phase) compose them to form one ontology. The goal is that modules are: 1) easier to reuse in other kinds of applications; 2) easier to build, maintain, and replace; 3) enable distributed development of modules over different locations and expertise; 4) enable the effective management and browsing of modules, e.g. enabling the construction of ontology libraries. In short, this approach to modularization is aimed with achieving scalable ontology engineering.

For automatic composition of modules, a composition operator can be used: all atomic concepts and their relationships (called lexons [JDM02, Mee99])) and all constraints, across the composed modules, are combined together to form one ontology (called modular ontology).

11.1 A Simple Example

In what follows, we give an example to illustrate the (de)composition of axiomatizations.² Figure 11.1 shows two ontologies for Book-Shopping and Car-Rental applications.³ No-

¹This section is short summary of the research on ontology modularization by Mustafa Jarrar, see [Jar05] for more details.

²In this section, the term ‘axiomatization’ is often interchanged with the term ‘ontology’ to mean the same thing.

³Please note that these modules are represented using the ORM [Hal01] graphical notation for simplicity and easy understanding. ORM (Object-Role Modeling) [Hal01] is a conceptual modeling method. ORM has an expressive and stable graphical notation since it captures many kinds of rules graphically (such as identity, mandatory, uniqueness, subsumption, subset, equality, exclusion, value, frequency, symmetric, intransitive, acyclic, etc.). Although ORM was originally developed as a database modeling approach, it has been also successfully reused in other conceptual modeling scenarios, such as ontology modeling [JDM02,

tice that both axiomatizations share the same axioms about “payment”.

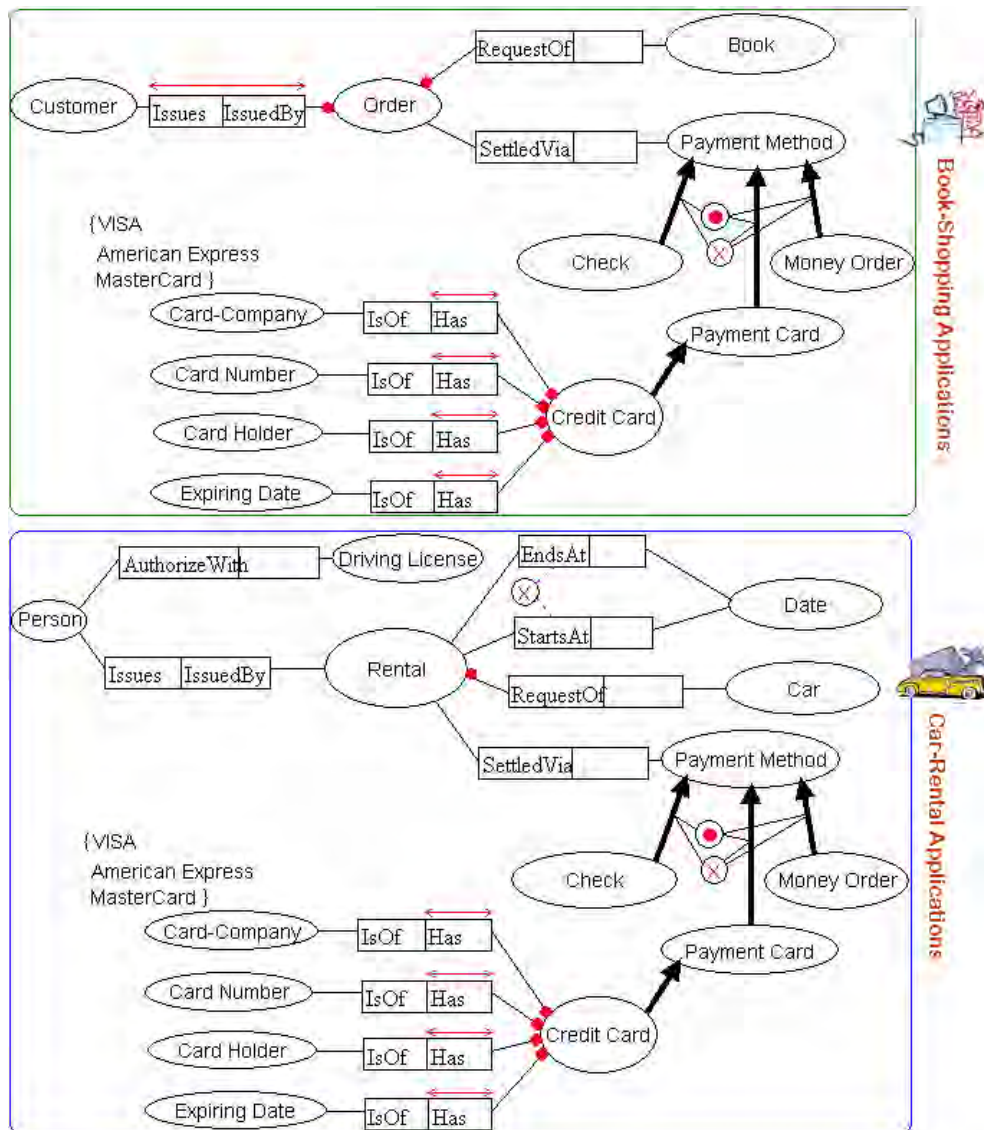


Figure 11.1: Book-shopping and Car-Rental axiomatizations.

Instead of repeating the same effort to construct the axiomatization of the “payment” part, the modularization principle suggests that we decompose these axiomatizations into three modules, which can be shared and reused among other axiomatizations (see Figure 11.2). Each application-type (*viz.* Book-Shopping and Car-Rental) selects appropriate modules (from a library of application axiomatizations) and composes them through a composition operator. The result of the composition is seen as one axiomatization.⁴

Jar05], business rule modeling language [Hal97, DJM02], XML-Schema conceptual design [BGH99], etc.

⁴The illustrated composition in this example is very simplistic, as each pair of modules overlap only in

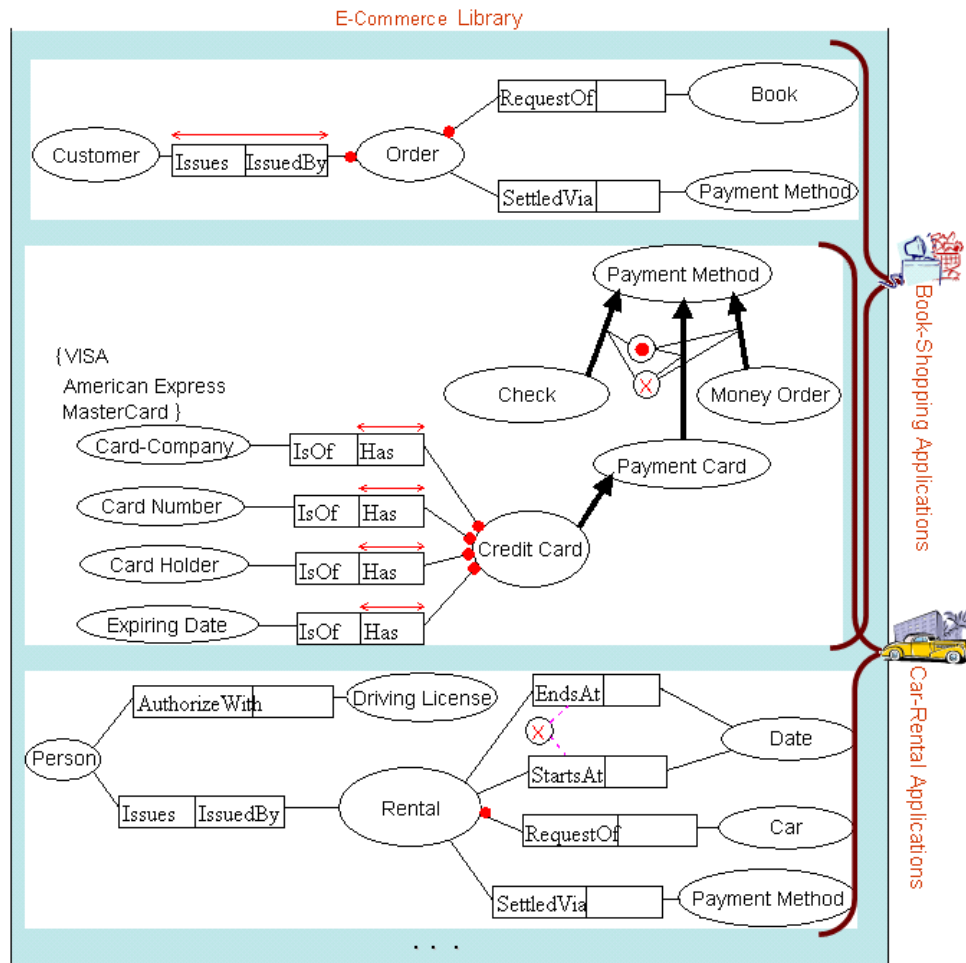


Figure 11.2: Modularized axiomatizations.

Developing and engineering axiomatizations in this way will not only increase their reusability, but also the maintainability of these axiomatizations, i.e. scalable and efficient ontology engineering. As the software engineering literature indicates, small modules are easier to understand, change, and replace [Par72, SWCH01]. An experiment by [BBDD97] proves that the modularity of object-oriented design indeed enables better maintainability and extensibility than structured design.

Decomposing axiomatizations into modules also enables the distributed development of these modules over different location, expertise, and/or stakeholders. As an analogy, compare the capability of distributing the development of a program built in Pascal with a program built in Java, i.e. structured versus modular distributed software development.

one concept, i.e. the “Payment Method”. In further sections, we discuss more complicated compositions, in which rules in different modules may contradict or imply each other.

11.2 Synthesis of Related Work

The importance of modularity has received limited attention within the knowledge representation community [SK03b]. Recently, as presented in this report, modularity has been adopted by some researchers to achieve more scalability for reasoning and inference services. A knowledge base is seen as a set of distributed knowledge bases, with each base referred to as a module. In this way reasoning is performed locally in each module, and the results are propagated toward a global solution. Global soundness and completeness (i.e. consistency) follows from the soundness and completeness of each local reasoner [WSC⁺04]. The performance of such reasoning is claimed to be linear in the tree structure in most cases. See Section 14 as an example of such approaches.

While such approaches are concerned with the modularity at the *deployment* phase of ontologies (i.e. distributed reasoning), Rector (see Section 12) has proposed another approach to modularity that is mainly concerned with the distributed *development* of the T-box of an ontology (i.e. scalable engineering). Rector's proposal is to decompose an ontology into a set of independent disjoint skeleton taxonomies restricted to simple trees. Disjoint taxonomies (i.e. modules) can then be composed using definitions and relationships between concepts in the different modules. In contrast to other approaches above, the result of such a composition can be seen as one local T-box. This approach is motivated by Guarino's analyses of types [G98]. Assuming that each type has a distinct set of identity criterion, when a type specializes another type, it adds further identity criterion to those carried by the subsuming type. The taxonomy of such types is always a tree.

11.3 Our Approach

In this section we introduce our approach to ontology modularization and composition on an abstract level. The formal and technical details will be provided in the following sections.

In our approach, we are mainly concerned with the modularity at the *development phase* of an ontology. Similar to Rector's proposal, our goal is to enable the "T-box" of an ontology to be developed as a set of modules and to later be composed to form one T-box.

However, unlike Rector's approach, we do not restrict a module to taxonomic relations between concepts. Modules are expected to include concepts, relations, and constraints (i.e. a typical T-box). In other words, we do not distinguish modules according to their level of abstraction, or according to the nature of their content. We call such distinctions as "ontology layering" or "ontology double-articulation", see [JDM02, JM02].

11.3.1 Modularity Criterion (Decomposition)

In what follows, we propose a modularity criterion aimed to help ontology builders to achieve *effective decomposition* and to guide them in why/when to release a part of an axiomatization into a separate module. The effectiveness of a decomposition can be seen as the ability to achieve a distributed development of modules and maximize both reusability and maintainability.

Subject: subject-oriented parts should be released into separate modules.⁵ For example, when building an axiomatization for university applications, one should separate between the financial aspects (e.g. salary, contract) and the academic aspects (e.g. course, exams). Encapsulating related axioms (on a certain subject) into one module will not only improve the reusability and maintainability of modules, but also enable the distributed development of modules by different people with a distinct expertise

Purpose: the general-purpose (or maybe called task-oriented) parts of an axiomatization could be released into separate modules. The notion of “general purpose” axiomatization refers to a set of axioms that are expected to be repeatedly used by different kinds of applications. For example, the axiomatization of “payment”, “shipping”, “person”, “address”, “invoicing”, is often repeated in many e-commerce applications. The reusability of such application axiomatizations is not based necessarily on their ontological foundation or abstraction levels but may be recognized simply from the experience of the creator and from best practices. For example, the wide adoption (i.e. repeatability) of the Dublin Core elements⁶ is based mainly on the simplicity of the encoding of descriptions (i.e. metadata) of networked resources.

Specific-purpose parts could also be modularized and released separately. In this way, the application-specificity of other modules will be decreased.

Stability: the parts that are expected to be frequently maintained or replaced could be released in separate modules. This affords other parts more stability and the unstable parts will themselves be easier to maintain and replace.

The criteria suggested above cannot be followed rigidly, as it is based on builders’ best practice and expectation of the reuse, maintenance, and distributed development of modules.

11.3.2 Module Composition

To compose modules we define a *composition operator*. All concepts and their relationships and all constraints, across the composed modules, are combined together to form

⁵This criteria is similar to, the so called “information hiding”, in software engineering, [Par72].

⁶<http://www.dublincore.org> (June 2004).

one axiomatization. In other words, the resultant composition is the union of all axioms in the composed modules.

As shall be discussed later, a resultant composition might be *incompatible* in case this composition is not satisfiable, e.g. some of the composed constraints might contradict each other.

Our approach to composition is constrained by the following consistency argument. An ontology builder, *when including a module into another, must expect that all constraints in the included module are inherited by the including module*, i.e. all axioms in the composed modules must be implied in the resultant composition. Formally speaking, the set of possible models for a composition is the intersection of all sets of possible models for all composed modules. In other words, we shall be interested in the set of models that satisfy all of the composed modules.

In Figure 11.3, we illustrate the set of *possible* instances (i.e. possible models) for a concept constrained differently in two modules composed together. Figure 11.3(a) shows a compatible composition where the set of possible instances for $M.c$ is the intersection of the possible instances of $M_1.c$ and $M_2.c$. Figure 11.3(b) shows a case of incompatible composition where the intersection is empty.

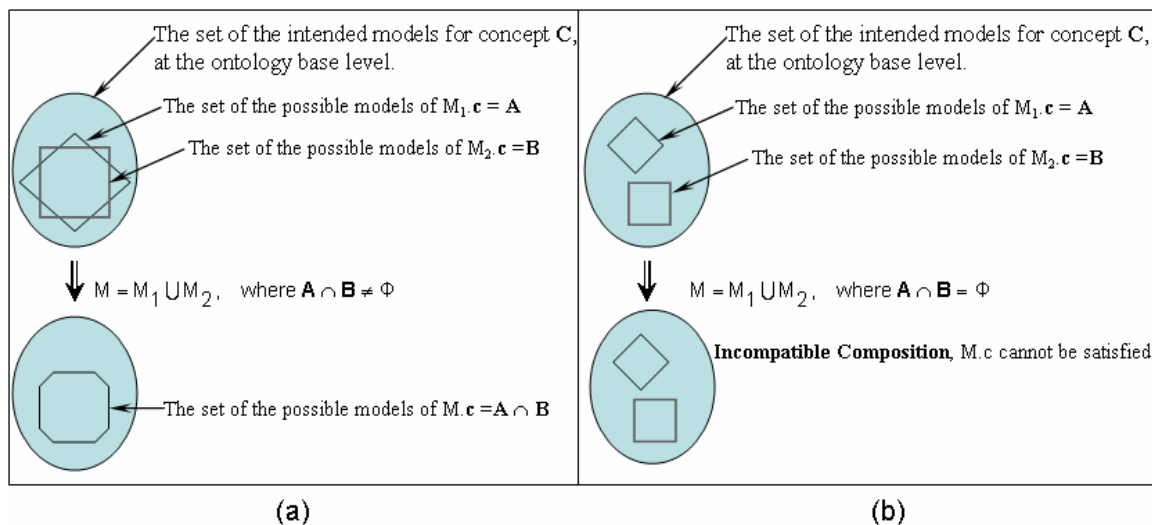


Figure 11.3: (a) Compatible composition, (b) Incompatible composition.

Notice that our approach to module composition is not intended to integrate or unite the extensions (i.e. A-boxes) of a given set of modules, as several approaches to ontology integration⁷ aim to do [SP94, SK03b, BS03]. Our concern is to facilitate ontology builders (at the development phases) with a tool to inherit (or reuse) axiomatizations *without "weakening" them*. In other words, when including a module into another module

⁷This might be seen as a designation between composition versus integration of ontological modules.

(using our composition operator, which we shall formalize in the next section) all axioms defined in the included module should be inherited by (or applied in) the including module.

11.4 Formal Framework

In this section, we introduce the formal framework of our approach to module composition. The approach is further illustrated by developing an algorithm for the automatic composition of modules specified in ORM.

11.4.1 Definition (Module)

A module is an axiomatization of the form $M = \langle P, \Omega \rangle$, where P is a non empty and finite set of atomic concepts and their relationships (we call it *lexon*⁸); Ω is a set of constraints which declares what should necessarily hold in any possible world of M . In other words Ω specifies the legal models of M .

11.4.2 Definition (Model, Module Satisfiability)

Using the standard notion of an interpretation of a first order theory, an interpretation I of a module M , is a *model*⁹ of M iff each sentence of M (i.e. each $\rho \in P$ and each $\omega \in \Omega$) is true for I .

Each module is assumed to be self-consistent, i.e. satisfiable. Module satisfiability demands that each lexon in the module can be satisfied [vBHvdW91]. For each lexon ρ in a given module M , ρ is satisfiable w.r.t. to M if there exists a model I of M such that $\rho^I \neq \emptyset$.

Notice that we adopt a strong requirement for satisfiability, as we require each lexon in the schema to be satisfiable. A weak satisfiability requires only the module itself (as a whole) to be satisfiable [Hal89, vBHvdW91].

11.4.3 Definition (Composition Operator)

Modules are composed by a composition operator, denoted by the symbol ' \oplus '. Let $M = M_1 \oplus M_2$, we say that M is the composition of M_1 and M_2 . M typically is the union of all

⁸A lexon is formed as $\langle T_1, r, r', T_2 \rangle$, where T refers to a Term (concept label), r refers to a role, r' refers to an inverse role, r and r' are the two parts of a binary relationship, for example $\langle \text{Customer, Issues, IssuedBy, Order} \rangle$.

⁹We also call it "legal model".

lexons and constraints in both modules. Let $M_1 = \langle P_1, \Omega_1 \rangle$ and $M_2 = \langle P_2, \Omega_2 \rangle$, the composition of $(M_1 \oplus M_2)$ is formalized as $M = \langle P_1 \oplus P_2, \Omega_1 \oplus \Omega_2 \rangle$.

A composition $(M_1 \oplus M_2)$ should *imply* both M_1 and M_2 . In other words, for each model that satisfies $(M_1 \oplus M_2)$, it should also satisfy each of M_1 and M_2 . Let $(M_1)^I$ and $(M_2)^I$ be the set of all possible models of M_1 and M_2 respectively. The set of possible models of $(M_1 \oplus M_2)^I = (M_1)^I \cap (M_2)^I$. A composition is called *incompatible* iff this composition cannot be satisfiable, i.e. there is no model that can satisfy the composition, or each of the composed modules.

In what follow we specify how sets of lexons and sets of constraints can be composed together.

Composing lexons

When composing two sets of lexons ($P = P_1 \oplus P_2$), a concept $M_1(T)$ in module M_1 and a concept $M_2(T)$ in module M_2 are considered exactly the same concept¹⁰ iff they are referred to by the same term T , and/or URI. Formally, $(M_1(T) = M_2(T))$. Likewise, two lexons are considered exactly the same $(M_1.\langle T_1, r, r', T_2 \rangle = M_2.\langle T_1, r, r', T_2 \rangle)$ iff $M_1.(T_1) = M_2.(T_1)$, $M_1.(r) = M_2.(r)$, $M_1.(r') = M_2.(r')$, and $M_1.(T_2) = M_2.(T_2)$.¹¹ See Figure 11.4.

In case that M_1 and M_2 do not share any concept between them (i.e. two disjoint sets of lexons), the composition $(M_1 \oplus M_2)$ is considered an incompatible operation¹², as there is no model that can satisfy both M_1 and M_2 .

Notice that in case concept is specified as “lexical” in one module and as “non-lexical” in another (e.g. ‘Account’), then in the composition, this concept is considered “non-lexical”. Lexical object types in ORM are depicted as dotted- ellipsis, Lexical vs. no-lexical in ORM is similar to DataProperty vs. ObjectProperty in OWL, or attribute vs. class in UML and EER.

Composing constraints

When composing two sets of constraints, first, all constraints need to be combined together ($\Omega = \Omega_1 \oplus \Omega_2$). Second, a satisfiability reasoning should be performed in order to find out whether the composition $(M = M_1 \cup M_2)$ is satisfiable. Finally, an optional step is to perform an implication reasoning to eliminate all implied constraints (also called “entailments”) from the composition.

In the first step, the *combination* of all constraints $(\Omega_1 \oplus \Omega_2)$ should be syntactically

¹⁰i.e. refer to the same intended models.

¹¹T refers to a Term (concept label), r refers to a role, r' refers to an inverse role, r and r' are the two parts of a relationship, example of a lexon $\langle T_1, r, r', T_2 \rangle$ is $\langle \text{Customer, Issues, IssuedBy, Order} \rangle$.

¹²In some practice cases, we weaken this requirement to allow the composition of disjoint modules. For example, in case one wishes to compose two disjoint modules and later compose them within a third module that results in a joint composition.

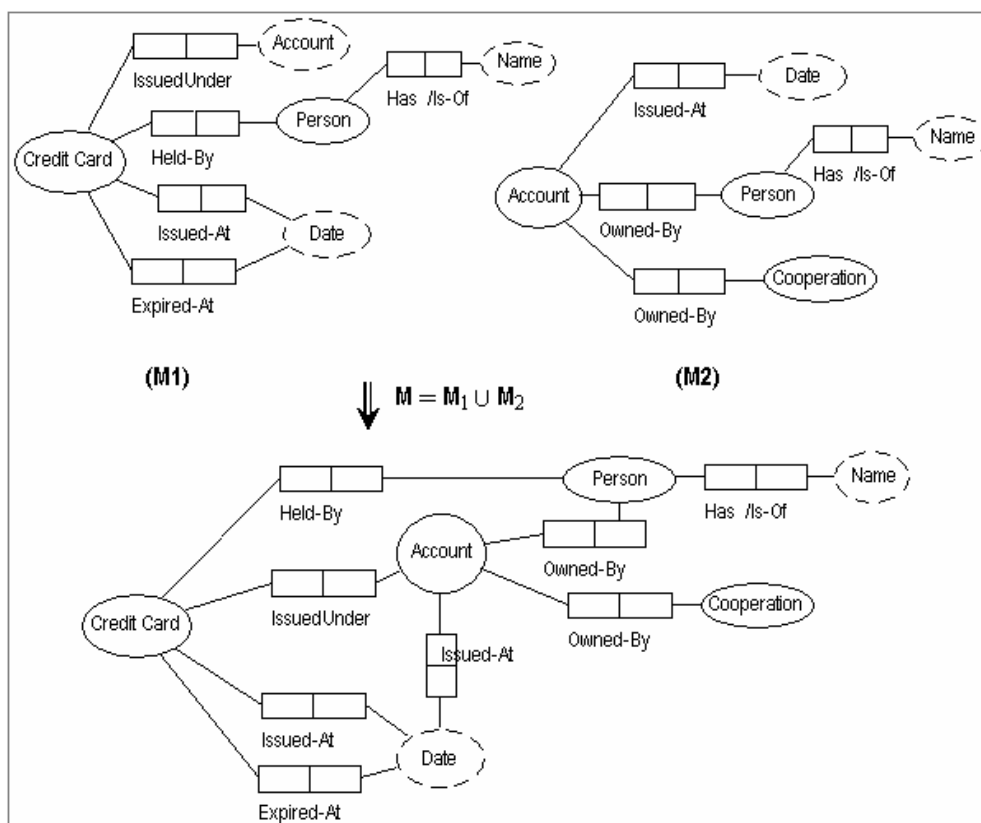


Figure 11.4: Combining ORM fact types.

valid according to the syntax of the constraint specification language. For example, some constraints need to be syntactically combined into one constraint. *The combination of a set of constraints should imply all of them.* To provide an insight into such combinations, in Figure 11.5, we show the combination of two UML cardinality constraints. Figure 11.6 illustrates several combinations of ORM constraints. Notice that in case of a constraint contradiction, the composition is terminated and considered an incompatible operation, as in Figure 11.6(d).

Figure 11.5: Combining UML constraints.

The ability to automate this process depends on the complexity of the constraint speci-

Figure 11.6: Examples of several combinations of ORM constraints: (a) combination of two value constraints, (b) combination of uniqueness, and frequency, (c) combination of subset and equality, and (d) combinations of equality and exclusion constraints.

fication language. In [Jar05] we have developed an algorithm - based on the above framework - to **automatically compose modules specifying in ORM**, this algorithm is also implemented in the DogmaModeler ontology modeling tool [Jar05]. Please see [JVM03] for real-life case study where this operator was successfully applied.

11.4.4 Definition (Modular Axiomatization)

A modular axiomatization $M = \{M_1, \dots, M_n, \oplus\}$ is a set of modules with a composition operator between them, such that $P = (P_1 \oplus \dots \oplus P_n)$ and $\Omega = (\Omega_1 \oplus \dots \oplus \Omega_n)$.

Notice that cyclic compositions are null operations, as the repetition of the same proposition has no logical significance. For example, the composition $M = ((M_1 \oplus M) \oplus M_2)$ equals $(M_1 \oplus M_2)$ and the composition $M = ((M_1 \oplus M_2) \oplus (M_2 \oplus M_1))$ also equals $(M_1 \oplus M_2)$.

11.5 Conclusion and Implementation

In this chapter we have presented an approach to modularize and automatically compose Ontology modules. This approach is fully implemented in DogmaModeler [Jar05], which is a software tool for modeling ontologies and business rules using the ORM graphical notation. DogmaModeler enables users to create, compose, add, delete, manage, and brows

ORM (modular) schemes. DogmaModeler also implements a library of ORM modular schemes, allowing different metadata standards (e.g. Dublin-Core, LOM, etc.) to be used for describing modules. This approach has been also used in a real-life case study (CCFORM EU project, IST-2001-34908, 5th framework.) for developing modular axiomatizations of customer complaints knowledge, see [Jar05, JVM03] for the experience and lessons learned.

Chapter 12

Engineering Robust Modules

by ALAN RECTOR, JEFF PAN

This chapter concentrates specifically on the engineering issues of robust modular implementation in logic based formalisms such as OWL. More specifically, we concentrate on the domain level ontology rather than the high abstract categories discussed by Guarino and Welty [GW00].

In our approach, the fundamental goal of modularity in the domain ontology is to support re-use, maintainability and evolution. The goal is only possible if:

- The modules to be re-used can be identified and separated from the whole.
- Maintenance can be split amongst authors who can work independently.
- Modules can evolve independently and new modules be added with minimal side effects.
- The differences between different categories of information are represented explicitly both for human authors' understanding and for formal machine inference.

12.1 Overview

We assume that the basic structure of the ontology to be implemented has already been organised cleanly by a mechanism such as that of Guarino and Welty, and that a suitable set of high level categories are in place. Our goal is to implement the ontology cleanly in as FaCT, OWL, or other logic-based formalism. Such formalisms all share the principle that the hierarchical relation is "is-kind-of" and is interpreted as logical subsumption – i.e. to say that "B is a kind of A" is to say that "All Bs are As" or in logic notation $\forall x. B(x) \rightarrow A(x)$. Therefore, given a list of definitions and axioms, a theorem prover or "reasoner" can infer subsumption and check whether the proposed ontology is self-consistent.

The list of features supported by various logic based knowledge representation formalisms varies, but in this chapter we shall assume that it includes at least:

- *primitive concepts* which are described by necessary conditions;
- *defined concepts* which are defined by necessary and sufficient conditions;
- *properties* which relate concepts and can themselves be placed in a subsumption hierarchy;
- *restrictions* which are constructed as quantified role-concept pairs, e.g. (restriction hasLocation someValuesFrom Leg) meaning "located in some leg";
- *axioms* which declare concepts either to be disjoint or to imply other concepts.

These mechanisms are sufficient to treat two independent ontologies as modules to be combined by definitions. For example, independent ontologies of dysfunction and structure can be combined in concept descriptions such as "Dysfunction which involves Heart" (Dysfunction and (restriction involves someValuesFrom Heart)), "Obstruction which involves Valve of Heart" (Obstruction and (restriction involves someValuesFrom (Valve and (restriction isPartOf someValuesFrom Heart))). In this way, complex ontologies can be built up from and decomposed into simpler ontologies. However, this only works if the ontologies are modular. The rich feature sets of modern formalisms such as OWL allow developers a wide range of choices in how to implement any given ontology. However, only a few of those choices lead to the desired modularity and explicitness.

Implicit information could be a problem – most frequently because either i) information is left implicit in the naming conventions and is therefore unavailable to the reasoner, or ii) information is represented in ways that do not fully express distinctions critical to the user. Amongst the distinctions important to users are the boundaries between modules. If each primitive belongs explicitly to one specific module, then the links between modules can be made explicit in definitions and restrictions as in the examples above. However, if primitive concepts are 'shared' between two modules, the boundary through them is implicit – they can neither be separated, since they are primitive, nor confidently allocated to one module or the other. Hence, it *matters* which concepts are implemented as primitives and which as constructs and restrictions. The key notion in our proposal is that modules be identified with trees of primitives and the boundaries between those trees identified with the definitions and descriptions expressing the relations between those primitives. We will discuss more details in the following sections.

12.2 Primitive Skeleton

We term that part of the ontology consisting only of the primitive concepts the "primitive skeleton". We term that part of the ontology which consist only of very abstract cate-

gories such as "structure" and "process" which are effectively independent of any specific domain the "top level ontology", and those notions such as "bone", "gene", and "tumour" specific to a given domain such as biomedicine the "Domain ontology".

The essence of our proposal is that the primitive skeleton of a domain ontology should consist of disjoint homogeneous trees.

1. The branches of the primitive skeleton of the domain taxonomy should form trees; i.e. no domain concept should have more than one primitive parent.
2. Each branch of the primitive skeleton of the domain taxonomy should be homogeneous and logical; i.e. the principle of specialisation should be subsumption (as opposed, for example, to paratomy) and should be based on the same, or progressively narrower criteria, throughout. For example, even if it were true that all vascular structures were part of the circulatory system, placing the primitive "vascular structure" under the primitive "circulatory system structure" would be inhomogeneous because the differentiating notion in one case is structural and in the other case functional.
3. The primitive skeleton should clearly distinguish:
 - (a) "*Self-standing*" *named concept*¹: most "things" in the physical and conceptual world – e.g. "animals", "body parts", "people", "organisations", "ideas", "processes" etc as well as less tangible notions such as "style", "colour", "risk", etc. Self-standing primitives should be *disjoint* and *open*, i.e. the list of primitive children should not be considered exhaustive (should not "cover" the parent), since lists of the things that exist in the world are hardly guaranteed exhaustive.
 - (b) "*Partitioning*" or "*Refining*" *named concept*: value types and values which partition conceptual (qualia - [GW00]) spaces e.g. "small, medium, large", "mild, moderate, severe, etc. For refining concepts: a) there should be a taxonomy of primitive "value types" which may or may not be disjoint; b) the primitive children of each value type should form a disjoint exhaustive partition, i.e. the values should "cover" the "value type". The exhaustive manner can naturally be represented by the use of datatypes. For instance, one can define that objects with size less than 10cm are small, that objects with size larger than 10m is large, and that anything between 10cm and 10m is medium.

In practice, we recommend that the distinction between "self-standing" and "partitioning" named concept be made in the top level ontology. However, in order to avoid commitment to any one top level ontology, we suggest only the weaker

¹The phrase "self-standing concepts" is problematic, but has so far produced less controversy than any suggested alternative. In Guarino and Welty they correspond to "types", "quasi-types" and certain concepts used to construct representation of "formal and material roles".

requirement for modularisation; i.e. that the distinction be made clear by some mechanism.

4. The range and domain constraints should never imply that any primitive domain concept is subsumed by more than one other primitive domain concept.

Note that requirement 2, that each branch of the skeleton be "homogeneous", does not imply that the same principles of description and specialisation are used at all levels of the ontology taken as a whole. Some branches of skeleton providing detailed descriptions (e.g. "forms and routes" of drugs or detailed function of genes) will be used only in specialised modules "deep" the ontology as a whole. Our proposal, however, is that when such a set of new descriptors is encountered, its skeleton should be treated as a separate module in its own branch of the skeleton.

The distinction between "self-standing" and "partitioning" named concept is usually straight forward and closely related to Guarino and Welty's distinction between "sortals" and "nonsortals" [GW00]. However, the distinction here is made on pragmatic engineering grounds according to two tests: a) Is the list of named things bounded or unbounded? b) Is it reliable to argue that the sub-concepts exhaust the super-concept? i.e. is it appropriate to argue that "Super & not sub₁ & not sub₂ & not sub₃... not sub_{n-1} implies sub_n"? If the answer to either of these questions is "no", then the concept is treated as "self-standing".

The first consequence of the criteria 1, 3 and 4 is that all multiple inheritances are inferred by ontology reasoners. Ontology authors should never assert multiple inheritances manually. The second consequence is that, for any two primitive self-standing named concepts, either one subsumes the other or they are disjoint. From this, it follows that any domain individual is an instance of exactly one most specific self-standing primitive concept.

A third set of consequences of criteria 1 and 3 is that a) declarations of primitives should consist of conjunctions of exactly one primitive and zero or more restrictions; b) every primitive self-standing concept should be part of a disjoint axiom with its siblings; and c) every primitive value should be part of a disjoint sub-class axiom with its siblings so as to cover its value type. Finally, criteria 4 limits the use of arbitrary disjointness and subclass axioms. Disjointness amongst primitives is permitted, indeed required by criterion 3.² Subclass axioms are allowed to add necessary conditions to defined concepts by causing them to be subsumed by further restrictions, but not to imply subsumption by arbitrary expressions containing other primitives.

²A stronger criterion concerning disjointness axioms is probably desirable. The only two use cases which we have seen which do not 'tangle' the ontology are a) disjointness between primitive siblings of a common parent; b) disjointness between existential restrictions to represent non-overlap in space, e.g. (has_location someValuesFrom Germany) disjoint (has_location someValuesFrom France).

12.3 Rationale

Minimising implicit differentia

This approach seeks to minimise implicit information. Not everything can be defined in a formal system; some things must be primitive. In effect, for each primitive, there is a set of implicit notions that differentiate it from each of its primitive parents (the Aristotelian "differentia" if you will). Since these notions are implicit, they are invisible to human developer and mechanical reasoner alike. They are therefore likely to cause confusion to developers and missed or unintended inferences in the reasoner. The essence of the requirement for independent homogeneous taxonomies of primitives is that there is exactly one implicit differentiating notion per primitive concept (i.e., different primitive skeleton don't share primitive concepts), thus confining implicit information to its irreducible minimum. All other differentiating notions must be explicit and expressed as "restrictions" on the relations between concepts.

Keeping the Skeleton Modular

The requirement that all differentiating notions in each part of the primitive skeleton be of the same sort – e.g. all structural, all functional – guarantees that all conceptually similar primitive similar notions fall in the same section of the primitive skeleton. Therefore our modularisation will always include notions that divide along natural conceptual boundaries. The requirement that the primitive skeleton of the domain concepts form primitive trees is very general and still requires ontology authors to make choices. For example, the notion of the "Liver" might be of a structural unit which serves a variety of functions. It might be classified as an "Abdominal viscera", "A part of the digestive system", or a part various biochemical subsystems. One such relationship must be chosen as primary – if we follow the Digital Anatomist Foundational Model of Anatomy [RSB98] or *OpenGALEN* [RR96], we will choose the simple structural/developmental notion that the Liver is an "Organ". All other classification will be derived from the description of the structure, relationships, and function of that organ. "Liver" will therefore be part of the organ sub-module of the structural anatomy module of the ontology.

Avoiding Unintended consequences of Changes

New definitions for new concepts can only add new inferences; they cannot remove or invalidate existing inferences. Likewise, adding new primitive concepts in an open disjoint tree can only add information. They may make new definitions and inferences possible, but they cannot invalidate old inferences (because the languages we consider are monotonic). Therefore definitions of new concepts and new disjoint concepts, or even entire disjoint trees, can be added to the skeleton with impunity.

There are three operations which can cause unintended consequences: i) adding new restrictions to existing concepts; ii) adding new primitive parents; iii) adding new unre-

Original Hierarchy	Normalised Skeleton Taxonomies																
Substance	...																
Protein	Substance																
'ProteinHormone'	Protein																
Insulin*	Insulin																
ATPase*	ATPase																
Steroid	Steroid																
'SteroidHormone'	Cortisol																
Cortisol																	
'Hormone'																	
'ProteinHormone'																	
Insulin*																	
'SteroidHormone'																	
'Catalyst'																	
'Enzyme'																	
ATPase*																	
	<p style="text-align: center;">Linking Definitions and Restriction</p> <table> <tr> <td>Hormone</td> <td> Substance & playsRole-someValuesFrom HormoneRole</td> </tr> <tr> <td>ProteinHormone</td> <td> Protein & playsRole someValuesFrom HormoneRole</td> </tr> <tr> <td>SteroidHomone</td> <td> Steroid&playsRole someValuesFrom HormoneRole</td> </tr> <tr> <td>Catalyst</td> <td> Substance & playsRole someValuesFrom CatalystRole</td> </tr> <tr> <td>Enzyme</td> <td> Protein & playsRole someValuesFrom CatalystRole</td> </tr> <tr> <td>Insulin</td> <td>→ playsRole someValuesFrom HormoneRole</td> </tr> <tr> <td>Cortiso</td> <td>→ playsRole someValuesFrom HormoneRole</td> </tr> <tr> <td>ATPase</td> <td>→ playsRole someValuesFrom CatalystRole</td> </tr> </table>	Hormone	Substance & playsRole-someValuesFrom HormoneRole	ProteinHormone	Protein & playsRole someValuesFrom HormoneRole	SteroidHomone	Steroid&playsRole someValuesFrom HormoneRole	Catalyst	Substance & playsRole someValuesFrom CatalystRole	Enzyme	Protein & playsRole someValuesFrom CatalystRole	Insulin	→ playsRole someValuesFrom HormoneRole	Cortiso	→ playsRole someValuesFrom HormoneRole	ATPase	→ playsRole someValuesFrom CatalystRole
Hormone	Substance & playsRole-someValuesFrom HormoneRole																
ProteinHormone	Protein & playsRole someValuesFrom HormoneRole																
SteroidHomone	Steroid&playsRole someValuesFrom HormoneRole																
Catalyst	Substance & playsRole someValuesFrom CatalystRole																
Enzyme	Protein & playsRole someValuesFrom CatalystRole																
Insulin	→ playsRole someValuesFrom HormoneRole																
Cortiso	→ playsRole someValuesFrom HormoneRole																
ATPase	→ playsRole someValuesFrom CatalystRole																

Figure 12.1: Normalisation of Ontology of Biological Substances and Roles.

stricted axioms.

Operation i) can be achieved either directly or by adding subclass axioms that cause one class to be subsumed by a conjunction of further restrictions. Adding new restrictions can be partially controlled by domain and range constraints on properties. If the ontology is well modularised, then the properties that apply to concepts in each section of the skeleton are likely to be distinct and therefore unlikely to conflict. The results for existential (*someValuesFrom*) restrictions are almost always easy to predict. They can only lead to unsatisfiability if a functional (single valued) property is inferred to have (i.e. "inherits") two or more disjoint values. Our experience is that in "untangled" ontologies this is rare and that when it does occur it is easily identified and corrected. The results for universal (*allValuesFrom*) and cardinality restrictions require more care but are at least restricted in scope by modularisation.

However, operations ii) and iii) are completely unconstrained. It is difficult to predict or control what effects follow. Hence the rules for modularisation preclude these constructs even though they are supported by the formalism.

12.4 Discussion

Examples & Relation to Other Methods

As a simple example consider hierarchy in Figure 12.1 for kinds of "Substances". The original hierarchy is tangled with multiple parents for items marked with '*' – "Insulin", and "ATPase". Any extension of the ontology would require maintaining multiple classifications for all enzymes and hormones. Modularisation produces two skeleton tax-

onomies, one for substances, the other for the physiologic role played by those substances. Either taxonomy can be extended independently as a module – e.g. to provide more roles, such as "neurotransmitter role", new kinds of hormone new kinds of protein or steroid, or entire new classes of substances such as "Sugars". The definitions (indicated by ' ') and restrictions (indicated by '→') link the two taxonomies. The resulting hierarchy contains the same subsumptions as the original but is much easier to maintain and extend. (To emphasise the point, the concepts defined in the modularised ontology are shown in single quotes in the original ontology.)

As a further illustration consider the independently developed ontology in Figures 12.2,12.3 adapted from Guarino & Welty (see [GW00] Figure 6). Figure 12.2 shows the initial taxonomy after Guarino and Welty's "Ontoclean" process. While ontologically clean, its implementation is significantly tangled. Figure 12.3 shows the same ontology untangled and modularised. Each of the changes makes more information explicit. For example, "Food" is classified in the original as part of the backbone simply as a kind of "Amount of matter". In the modularised ontology in Figure 12.3, the relation of "Food" to "EatenBy Animal" is made explicit (and the notion of "plant food" therefore explicitly excluded a decision which might or might not be appropriate to the application but which would likely have been missed in the original. Note also that the nature of the relationship between "red apple" and "red", "big apple" and "big", is now explicit.

The relationship between "lepidopteran", "Butterfly" and "Caterpillar" which causes Guarino and Welty some difficulty as an example of "phased sortals" poses no problem; the relationship of each entity to the generic and to the phase is explicit. Furthermore, general notions such as "group" have been represented explicitly in a re-usable form and ambiguities addressed, e.g. Was "group of people" intended as a group *only* of people, or *at least* of people? Need a group have any members at all? The modularised representation forces the choice to be explicit rather than leaving it to the individual interpretation of the linguistic label.

Experience

Experience and several experiments support our contention that these techniques are a major assistance in achieving the goals set out in the introduction – *explicitness* and *modularity* in order to support *re-use*, *maintainability* and *evolution*.

This approach has been used throughout *OpenGALEN* and related ontologies over a period of fifteen years [Rec99]. In fact, many of the features of *GRAIL*, the formalism used in *GALEN*, were designed around these precepts [RBG⁺97]. Throughout this experience we have found no situation in which the suggested modularisation could not be performed. The requirement to limit the primitive skeleton to simple disjoint trees may seem restrictive, but it does not actually reduce expressiveness. In our experience, violation of this principle almost always indicates that tacit information is concealed which makes later extension and maintenance difficult. Furthermore, this approach has proved easy to explain to new ontology developers and has been one of the key strategies to sup-

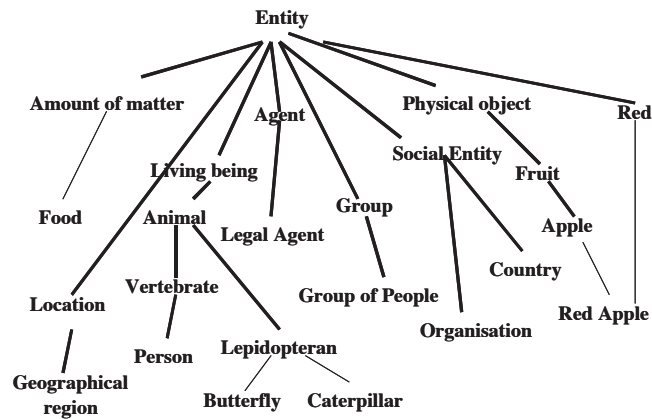


Figure 12.2: Example ontology from Guarino & Welty.

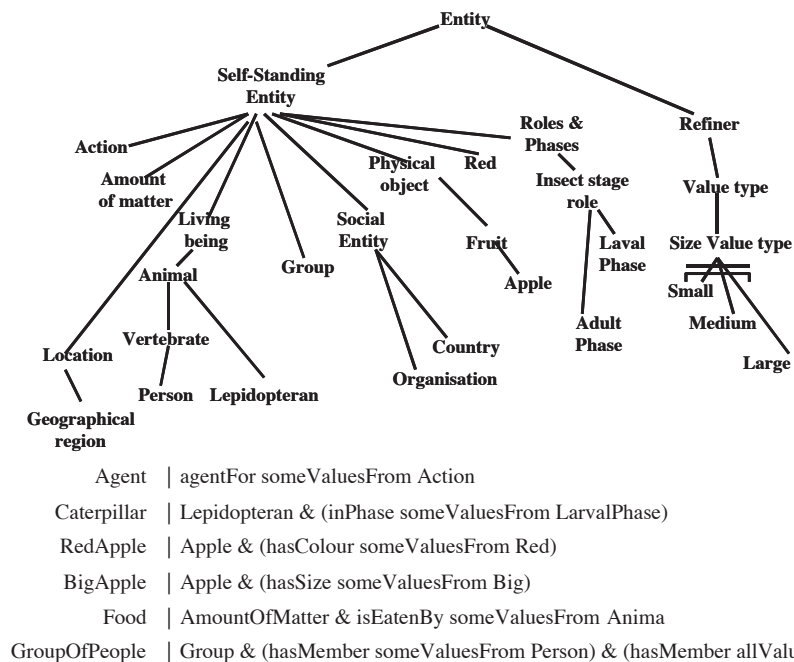


Figure 12.3: Untangled skeleton for example ontology 12.2 plus definitions linking independent branches.

port loosely coupled development [RZS⁺99]. Interestingly, Gu and her colleagues have independently proposed *post hoc* decomposition into disjoint trees as a means to improve maintainability of large ontologies represented in frame systems with multiple inheritance [GPG⁺99].

We have no comparative data on effort for maintenance, but the combination of modularisation and the use of intermediate representations [RWRR01, RZS⁺99] has allowed us to develop and maintain a large ontology (~30,000 concepts) in a loosely coupled cooperative team consisting at times of up to nine centres in seven countries. The central maintenance and integration effort has been reduced to roughly ten per cent of the total. New modules, for example for methodology and equipment for non-invasive surgery, have been added without incident, almost without comment – e.g. it was possible to add the notion of an "endoscopic removal of the gall bladder/ appendix/ ovary/ ulcer/..." in numerous variants to account for different countries' differing practices without any change the modelling of "removal of gall bladder/ appendix/ ovary/ ulcer/...".

Further evidence for the effectiveness of modularity comes from a study comparing the manually organised UK classification of surgical procedures from Clinical Terms Version 3 (CTv3) with corresponding parts of *OpenGALEN* [RPR⁺98]. One source of discrepancies was the inconsistent use in CTv3 of "removal" and "excision" – in some cases removals of a structure were classified as kinds of excisions of the same structure; in others the reverse (i.e., excisions were classified as kinds of removals of the same structure). In *OpenGALEN* because ontology is modularised, and "excision" and "removal" are primitives in a module separate from the anatomic structures removed or excised, the same policy is automatically maintained throughout. To take a second example from the same study, another set of discrepancies was traced to minor differences in anatomical boundaries reflecting genuine differences between experts. Each change to the anatomical module in *OpenGALEN* could be done in a single place in the anatomy module. Each corresponding change in CTv3 required changes to every surgical procedure concept affected and were widely distributed throughout the surgical procedure model.

Another evidence for the approach comes from the re-use of the *OpenGALEN* ontology as the basis for the drug information knowledge base underlying the UK Prodigy project [SWRR99]. Perhaps the most dramatic example of the methodology was work on the "simple" problem of forms, routes of administration and preparation of drugs. Although there are only a few hundred concepts, they are densely interconnected and classification had resisted concerted efforts by standards bodies for over two years. Restructuring the classification as a modularised ontology solved the problem in weeks [WC01].

12.5 Issues and Problems

The Notion of "self-standing"

The notion of "self-standing concept" can be troublesome. In most cases it corre-

sponds to Guarino and Welty's notion of "sortal"; in a few there are questions. For example, consider 'colour'. On the one hand, 'colours' could be considered as partitioning a "qualia space", and the notion of an "identity condition" for colours is problematic. However, in practice, the list of named colours is indefinitely large and constantly growing – witness the efforts of paint companies and interior decorators. To claim a closed list would therefore be inappropriate in most contexts. It is a rare context in which one would be confident in saying "If it is not red or yellow or blue or green... then it must be [say] brown". For most ontologies, we therefore suggest treating colours as "self-standing". By contrast, in most contexts we would be happy to accept that "If a measurement is neither low nor normal then it must be elevated". This is true even though we might provide intensifiers such as 'very' or an alternative partition that included "sky high" and "rock bottom". Hence in most ontologies we would recommend that such "modifiers" be treated as "partitioning".

Meta-knowledge

A better solution might be argued to be to make the notion of "self-standing" and "partitioning" meta knowledge. These notions are really knowledge about the concepts rather than about all of their instances. Likewise, the notion of whether a concept ought to be part of the primitive skeleton, might be better expressed as meta-knowledge. *OpenGALEN* and *OWL-DL* both exclude meta-knowledge within the language. Although it is permitted in *OWL full*, the reasoning support is ill defined. Implementing the distinctions in the ontology itself as suggested here might be considered to be an engineering "kluge" to cope with the limitations of DL classifiers. We would accept this point of view while maintaining the importance of the distinction itself. Hence we advocate that the criterion for modularisation be that there is a means for distinguishing between "self standing" and "partitioning" concepts without specifying the method by which the distinction be made. (A full discussion of the role of meta-knowledge in ontologies for the Semantic Web and the *OWL* family of languages is beyond the scope of this paper.)

12.6 Conclusion

To sum up, one can treat modules in complex ontologies as (smaller) ontologies. The ability of logical reasoners to link independent ontology modules to allow them to be separately maintained, extended, and re-used is one of their most powerful features. However, to achieve this end sufficient information should be explicit and available to both reasoners and authors. The large range of options provided by Description Logics mean that implementers need guidance on to achieve this end. The approach presented in this chapter is based on fifteen years' experience in the development of large (>35,000 concept) biomedical ontologies. The procedures are not an absolute guarantee of a clean, untangled implementation. Not all obscure constructs are completely debarred nor all unintended consequences eliminated, but they are greatly reduced. Others may wish to

challenge these criteria or propose further restrictions. However, we believe that if the potential of OWL and related DL based formalisms is to be realised, then such criteria for modularisation need to become well defined.

Chapter 13

A Contextualization Mechanism for Modularization

by CHRISTINE PARENT, STEFANO SPACCAPIETRA

Database researchers have since long practiced or investigated ways of modularizing a database schema that are similar to the ontology modularization discussed in this deliverable. The database approach to modularization is twofold, either application-oriented or data structure-oriented. The application-oriented approach, somehow similar to the proposal by Jarrar (cf. Chapter 11), consists in partitioning the database schema (roughly equivalent to the T-box in description logics) into smaller subsets, easier to grasp for humans. Each schema subset, called module, describes the subset of the world of interest that corresponds to one task or functionality of the application. For instance, when designing the schema of the GESREAU database for water resources management for the Vaud canton (Switzerland), application designers asked to partition it into eight modules: Hydrographic network, Measuring stations, Resource management, Fauna, and so on. Usually, the modules composing a database schema are disjoint. Usually too, instances are not taken into account in the process; modules are used only for designing purposes.

The data-structure oriented approach, somehow similar to the partitioning approach presented in Chapter 10, aims at reducing the size of schema diagrams (in terms of number of object and relationship types), so that a schema can be easily displayed (e.g., in CASE tools for database design) and easily understood by humans (because of reduced complexity). These approaches result in a multi-level schema definition, where the most synthetic level shows a kind of summary of the original schema (still structured as a normal schema), such that each component is either a component of the original schema or is a representation for an underlying "module". User can iteratively examine the content of each "module", down to the level of the original detailed schema. Modularization in this sense consists in defining rules for composing elementary "modules" from the existing schema elements, such that each "module" can be replaced by a single, higher-level object or relationship type. For example, a simple modularization rule consists in creating a "module" for each is-a hierarchy of object types, and denoting the "module" in the higher

levels only by its root object type.

More recently, context-dependent information management has been emphasized as a means to adapt the description of databases, web services, or user profiles, to a specific perception, or context. In this approach, contrarily to classic schema modularization, each element of a data description may have several different representations, and each instance may have several different values according to context. In particular, an European IST project, MurMur, has been devoted to the specification and implementation of a framework that supports context-aware databases, or, more precisely and using the terminology of the MurMur project, that supports multiple perceptions and multiple representations for databases in general as well as for geographical databases [PSZ05]. The MurMur framework consists in a data model, called MADS, which is an extended spatio-temporal entity relationship data model, and its associated manipulation and query languages. The data model and languages have been implemented on top of existing database management systems (DBMS) and geographical information systems (GIS).

A MADS multi-contexts (also called contextual) database is defined for several, say N , contexts, and contains the equivalent of N classic mono-context databases, obeying usual consistency rules, plus a number of implicit and explicit links among the mono-context databases. This contextual database, taken as a whole, does not necessarily obey the usual consistency rules for classic mono-context databases. It can contain different (one per context) representations for the same fact that may be conflicting at the schema or at the instance level. It may be, for instance, that for one context $C1$ the entity type A subsumes the entity type B , and that the subsumption does not hold in another context $C2$. Or, in context $C1$ the entity types A and B are related by a part-of relationship, and they are not linked in the context $C2$. As a last example, the instance $i0$ of the entity type A has different values in contexts $C1$ and $C2$. This potential for inconsistency is also outlined in the contribution in Chapter 14. However, the data manipulation languages allow users to correctly manipulate the database in one of two modes: mono-contextual database or multi-contextual database. In the second mode, the user sees several representations, can access any one of them, and can navigate from one representation to another one.

As MurMur results are valid for any data model, be it semantic, relational, or object-oriented, we believe that they can also be used for the definition and management of context-aware ontology services. Therefore, in this chapter we propose to adapt to ontology modularization the ideas from the MurMur project, whose main results may be summarized as follows:

- The development of an extended entity relationship data model supporting multiple representations depending on the context. The extension relies on two meta-concepts, the stamp that identifies a context, and the representation that is the description of an element of the schema for a given context, or the value of a database element for a given context.
- A set of rules defining the consistency of the representations in a contextual data-

base.

- Extended data manipulation and query languages allowing users to insert, update, remove, and access context dependent representations of instances.
- A set of rules restricting the access rights of users to the representations corresponding to the contexts they are allowed to use.

The sequel of this chapter describes these results with more details. The basic initial assumption of the MurMur approach is that each context is simply identified by a name (as in C-OWL [BGvH⁺03, BGv⁺04]), and these names are used to express to which contexts a given representation belongs (we say we "stamp" facts with context names, denoted s_1 , s_2 , etc.). At this stage, we abstract from the semantics of contexts as well as from the possible semantic relationships that may hold between contexts. Examples of possible semantics of contexts include the personal and subjective viewpoint of the designer, the semantic resolution required by an application, and the spatial resolution for spatial data. Examples of possible semantic relationships between contexts are the assertion that context s_1 includes context s_2 , i.e., every representation that belongs to s_1 also belongs to s_2 , and the assertion that context s_3 is equal to the union of contexts s_4 and s_5 .

13.1 Stamping

From data definitions (metadata) to data values, anything in a database relates to one or several contexts. The first step for the database administrator is to identify the contexts that are to be supported by the database and to associate a unique stamp to each one of them. This defines the set of stamps that are allowed for use with the database. We say that the database schema is stamped with this set. For instance, for a risk management application we used as test case in the MurMur project, the application designer identified nine contexts based on the combination of user profile (either managers, in charge of decision-making processes, technicians, in charge of observations, measurements and risk map preparation, or general public) and three spatial resolution levels.

Stamping an element defines for which contexts the element is relevant. Thus, an element that has a single representation may bear multiple stamps, meaning that the same representation is shared by several contexts. Representation consistency mandates that stamps associated to an object (or relationship) type form a subset of the stamps associated to the schema. Similar rules apply to properties within a type. An object or relationship type relevant to several contexts may show different properties depending on the context. Consequently, a property may be stamped with a subset of the stamps associated to the type it belongs to. This is illustrated in Figure 13.1. The same applies at the value level. A multi-context attribute may have different values that are specific to given contexts.



Building	s1  s2 
s1, s2	
s1, s2: number : (1:1) integer s1: usage : (1:1) enum (residential, commercial, public/administrative, other) s2: usage : (1:1) enum (residential, non-residential, other) s1, s2: description : (1:1) string f(P) s1: entrancePoint : (1,n) ● s1: owner : (1,1) string s2: height : (0,1) integer s1: constructionDate : (1,n) ⊕ s2: constructionDate : (1,1) ⊕	
s1, s2: KEY number	

Figure 13.1: An illustration of a context-varying object type, bearing stamps s1 and s2.

For instance, in multilingual databases, a property such as **riverName** may take different values according to the language in use within the context (e.g., Rhin, Rhein, and Reno).

Complementarily to stamping database elements, transactions accessing the database should be given a means to specify which contexts (one or many) they adhere to, which determines which representations (data types or values) are relevant to them. We assume that transactions issue an **"OpenDatabase"** command to specify which contexts (stamps) they want to use. Matching this set with the sets of stamps associated with the object and relationship types defines which object and relationship types are actually visible to the transaction, and with which properties. Thus, stamping provides functionality similar to a subschema definition capability, with the advantage that this approach maintains an integrated view of all contexts, while subschema definition (as provided in CODASYL-like database systems) isolates each schema definition.

13.2 Multiple Representations Modeling

Each element of the schema (object type, relationship type, attribute, method) and each element of the database (set of instances of an object or relationship type, object instance, relationship instance, attribute value) may have different representations according to the context. For example, at the schema level, an object type may have different descriptions (representations) according to the context; at the database level, an object type may have different sets of instances, an instance may have different values according to the context.

Each representation, pertaining to the schema or database level, participates in one or several contexts. Stamping provides an easy way to identify representations that belong to a given context. Figure 13.1 illustrates the case of an object type holding two different representations of buildings, one for context s1 and another one for context s2. Buildings are spatial objects (i.e., objects whose spatial extent is relevant for the applications). Depending on resolution, the spatial extent is represented either as a complex area (more precise description, stamp s1) or as a simple area (less precise description, stamp s2). Icons on the right hand side of the object type name show the existence of the alternative geometries. The list of attributes shows that:

- **number** is a monovalued and mandatory (minimum and maximum cardinalities 1:1) attribute shared by s1 and s2. It serves as key for the object type.
- **usage** is a monovalued mandatory attribute in both s1 and s2, with enumerated domains specific to each representation.
- **description** is a shared monovalued mandatory attribute whose value is a function of stamps. We call this a context-varying attribute, identified as such by the f(C) notation. For instance, the same building may have a different description for s1, e.g. "Individual house", and for s2, e.g. "Private house".
- **entrancePoint** and **owner** are mandatory attributes that only belong to representation s1.
- **height** is a monovalued optional attribute that only belongs to representation s2.
- **constructionDate** is a mandatory attribute in both s1 and s2, but it is monovalued for s1 and multivalued for s2.

We say that **Building** is a context-varying object type, as the actual representation of building objects changes from one context to another. **Building** is both multi-representation (it holds two representations) and multi-context (it relates to two contexts). At the instance level, the fact that two representations relate to the same real world entity is in this case conveyed by the fact that the two representations share the same oid (we refer to this as an implicit link between contexts). Logically they are part of the same object instance.

Another way to organize alternative representations for the buildings is to define two separate object types, e.g. **Building** and **IGNBuilding**, each one holding the representation for the corresponding context. The two object types are then explicitly linked by a relationship type that holds a specific inter-representation semantics (cf. Figure 13.2), expressing that two linked instances describe the same building. Instances of the relationship type (**Correspond**, in Figure 13.2) tell which object instances represent the same buildings. The inter-representation semantics is visually indicated on schema diagrams by the \Leftrightarrow icon. In Figure 13.2, the cardinalities of **Correspond** show that buildings that

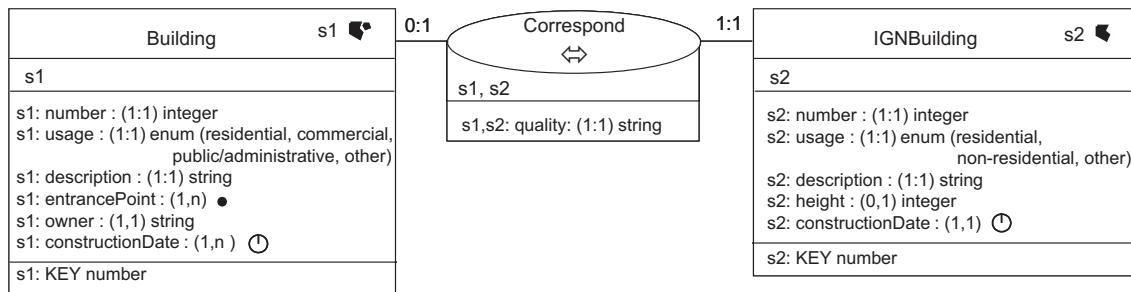


Figure 13.2: Modeling buildings as two mono-context object types linked by a multi-context inter-representation relationship type.

have a representation at the most detailed level, s_1 , do not necessarily have one at the less detailed level, s_2 .

The mapping between the instances that represent the same real world entities for different contexts is not always a 1:1 mapping. It can be 1:n or n:m. Moreover, in some cases the mapping may relate not individual instances but groups of instances. This happens when the object types represent the real world entities not as wholes but through their components. In order to support these cases, the MADS model offers a new kind of relationship type, called multi-association relationship type. Contrarily to a classic relationship, each role of a multi-association relationship links a set of instances.

Inter-representation relationship types are not transitive: It is possible to have an inter-representation relationship type between object types O_1 and O_2 and one between object types O_2 and O_3 without having one between O_1 and O_3 . Thus, if more than two types are used to describe and store the desired set of representations for a given set of entities, these types have to be linked by as many inter-representation relationship types as appropriate. Consider, for instance, a database with object types **Person**, **Company**, and **CarOwner**, and rules for this database stating that persons and companies are separate sets of objects, while both persons and companies may own cars. In this case there will be an inter-representation relationship type between **Person** and **CarOwner**, another one between **Company** and **CarOwner**, but none between **Person** and **Company**.

The two solutions for describing multiple representations are not always interchangeable. The first solution, a unique object type grouping two (or more) representations, requires that the instances belonging to each context be interrelated by a (total or partial) 1:1 mapping.

Instances are another component of a type; hence they obey the same stamping consistency rule as properties. In our example, a **Building** instance can be created by a transaction holding stamp s_1 , or holding stamp s_2 , or holding both stamps. If the transaction holds both s_1 and s_2 , it may create the whole value of the **Building** instance, as in a normal INSERT operation in traditional databases. If the transaction holds only one stamp, it can only create the representation corresponding to this stamp. In short, it can create only part of the instance. This straightforwardly generalizes to transactions holding

a subset of the set of stamps associated to a type. Partial creation of an instance means that two transactions using different stamps must be able to share an identification mechanism (e.g., the building number) guaranteeing that their data can correctly be merged by the DBMS into a single instance. Consider the creation of a **Building** instance. It can be done by two transactions. The first one creates a new instance for a specific context, say *s1*, like in the following insert operation. This transaction has to provide a value for all mandatory attributes of the *s1* representation.

```
INSERT INTO Building VALUES (
  stamp = {s1},
  geometry = complexarea { {(x1,y1), ... (xn,yn)} },
  number = 5001,
  usage = "residential",
  description = "Individual house",
  entrancePoint = { (a1,b1) },
  owner = "Dupont",
  constructionDate = { 1980, 1999 })
```

Then a second transaction may add a new representation to the instance. For example, the following operation adds an *s2* representation to the previously created building instance:

```
ADDREP TO Building WHERE number = 5001 VALUES (
  stamp = {s2},
  geometry = simplearea {(x'1,y'1), ... (x'n,y'n)},
  usage = "residential",
  description = "Private house",
  height = 12.50,
  constructionDate = 1980)
```

Stamping instances also allows determining which instances are visible to a transaction. A transaction with stamp *s1* will only see the instances stamped *s1* or (*s1*, *s2*). Similarly for transactions stamped *s2* only. A transaction with both stamps *s1*, *s2* will see all **Building** instances, but it will still have to be aware that the actual format of each instance varies according to its stamps.

13.3 Context-Varying Relationship Types

Similarly to object types, relationships may be context-varying. Context stamps may be associated to a relationship type, to its attributes, methods, and population. Its structure (e.g., participating roles and their cardinalities) and semantics (topology, synchronization, aggregation...) may also be context-varying. However, the objects (or sets of objects) involved in role of a relationship instance cannot change from one context to another one within the same relationship instance. This is because the linked objects are inherently part of the relationship instance, i.e. they participate in the identification of the relationship instance. If any of them is replaced with another object, it is not anymore the same relationship instance. So, for a given relationship instance and role, it is always the same



Figure 13.3: A stamped (topological) relationship type.

object instance that is linked, whatever the context is (in case of a multi-association, for a given instance of the multi-association and for a given role, it is always the same set of instances that are linked, whatever the context is). On the other hand, a context may see only a subset of the roles, but always at least two roles, as each context must always provide a consistent database.

A usual rule regarding relationships says that pending roles are not allowed. Hence, access rules have to guarantee that a relationship type is visible to a transaction only if for at least two of its roles the linked object types are visible, so that a consistent unit of information can be delivered to the transaction. The same rule applies at the instance level: Only visible instances of the relationship that link object instances visible to the transaction may be delivered to the transaction. Consider, for instance, the **Correspond** relationship type in Figure 13.2. As it links two object types bearing different stamps, transactions willing to see it must hold both $s1$ and $s2$ stamps. Having these two stamps also gives visibility over the relationship type, which bears stamps $s1$ and $s2$. However, relationship types do not need to be relevant for all contexts of the linked object types. For instance, Figure 13.3 shows a **GivesAccess** relationship type that only bears the stamp $s1$, whereas the linked object types have stamps $s1$ and $s2$. Consequently, transactions holding only stamp $s2$ do not see **GivesAccess** (they ignore which instances of **Building** are linked to instances of **Road**, and vice versa).

In fact, the way relationship types are stamped does not depend on how the linked object types are stamped. It may be perfectly correct to have the **Correspond** relationship type in Figure 13.2 stamped with a stamp $s3$. In this case, the relationship type would be only visible to transactions holding the three stamps, ($s1, s2, s3$).

13.4 Context-Varying Is-a Links

Is-a links, like the other concepts of the data model, are stamped. The population inclusion and inheritance constraints that characterize is-a links mandate that the management of multiple representations obeys these constraints. In a multi-context framework, this translates into the constraint that an is-a link must belong to the same context as the object types it links. In other words, the super-type and the sub-type must share one or several stamps, and the is-a link is stamped with a non-empty subset of these common stamps. For instance, the is-a links shown in Figure 13.4 are only visible for context $s1$.

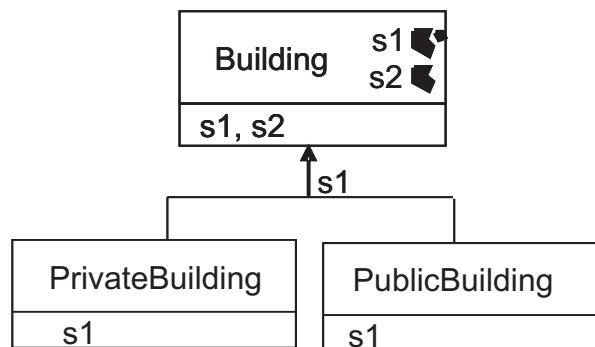


Figure 13.4: Multiple representations supported via is-a links.

13.5 Context-aware Querying

As already stated, transactions accessing a multi-represented database have to specify which contexts they adhere to, so that consistency in the use of data can be monitored. Before accessing data, a transaction has to issue an **OpenDatabase** operation (any syntax used in this section is simplified and for explanatory purposes only):

OpenDatabase (database-name, set-of-stamps)

The operation specifies the stamps the transaction wants to use. This operation restricts the view of all operations in the transaction to the database elements that have at least one of the quoted stamps, i.e. to the representations defined by the set of stamps **set-of-stamps**. If **set-of-stamps** includes only one stamp, the transaction sees a mono-context database, similar to traditional databases. If **set-of-stamps** includes more than one stamp, what the transaction sees is something like the corresponding collection of mono-context databases, but with possibly integrated specifications (i.e., context-varying types) and augmented with inter-contexts links (including inter-representation relationships), only visible to this kind of transactions.

Having completed an **OpenDatabase** operation, any visible representation can be accessed via selection operations. Because of the multi-representation framework, which representations are requested should be specified without ambiguity in the operation.

Assume, for instance, that a transaction T1 has opened, using stamps s1 and s2, a database that includes the **Building** object type as shown in Figure 13.1. T1 now issues the following selection query:

(1) Selection [number = 5001] Building

This query selects the **Building** instance bearing **number** 5001. First, as the transaction holds the two stamps that are attached to **Building**, and **Building** is not stamped in the query, all **Building** instances are visible to the query, with both s1 and s2 representations. Moreover, as the **number** attribute has a unique representation, common to both contexts, s1 and s2, there is no need to qualify **number** with a context stamp. Finally, as

the query sees both representations, the resulting instances have the same format as the one they have in the database, i.e., their formats are heterogeneous: s_1 , s_2 , or (s_1+s_2) formats.

A query may also restrict the representations it wants to see by explicitly stating the desired stamps. For instance, if the transaction T1 now issues the query:

(2) Selection [number = 5001] Building.defAtStamp(s1)

This query only returns **Building** instances that are stamped with at least s_1 and satisfy the predicate on **number**. The notation `Building.defAtStamp(s1)` uses a method defined for context-varying types. The method `defAtStamp()` restricts the definition of the **Building** object type to the one that holds for the stamp specified by the parameter (here, s_1). Instances are returned in the s_1 format: The schema of the resulting object type only shows attributes from the operand type that exist for stamp s_1 , with their s_1 definition and value.

Let us now consider a transaction T2 that has opened the same database with the representation stamp s_1 only. Then the following T2 query has the same result as query (2):

(3) Selection [Number = 5001] Building

When a query accesses an attribute (or method) that has a different definition according to the context in use, it has to specify which representation it wants to see, otherwise the query would be ambiguous. For instance, a query of transaction T1 that wants to select the buildings such that the value of **usage** is "commercial" should specify which representation of **usage** should be used, as in the following operation:

(4) Selection [usage.defAtStamp(s1) = "commercial"] Building

This operation returns instances whose format complies with s_1 or $(s_1$ and $s_2)$ stamps. Instances stamped s_2 only are eliminated, as the predicate cannot be checked. Note that associating the stamp to **Building**, instead of **usage**, would return the same instances but in format s_1 only.

Similarly, when accessing an attribute whose value varies according to the context, queries have to specify which value (which representation) they want. For instance, a query to retrieve buildings whose description for stamp s_1 is "Individual house" should be written with the s_1 stamp associated to the value. To this purpose the method `atStamp(s)` has been defined for context-varying types: It selects the value of the attribute that holds for the stamp s .

(5) Selection [description.atStamp(s1)g"Individual House"] Building

A query may also want to search for a value, whatever the context is. Then it has to use an existential or universal quantifier to define the intended semantics of the query. For instance, the following two queries yield different results. :

(6) Selection [$k_s \in D_{stamp}$ description.atStamp(s)="Individual House"] Building

(7) Selection [is∈Dstamp description.atStamp(s)='Individual House'] Building

The first one selects buildings that bear **description** "Individual house" for at least one of their representations (**Dstamp** is the set of context stamps of the database). The second one requires that **description** is equal to "Individual house" for all representations.

13.6 Conclusion

The multi-representation approach of the MurMur project aimed at supporting a global database containing a set of classic mono-context databases, such that elements that describe different representations of the same set of real world phenomena can be related. Two corresponding elements can be either merged in a multi-context object (or relationship) type. Or they can be described each one by an object type, and the two object types be related by a peculiar relationship type whose semantics is "inter-representation". A contextual database comes with a specific, additional consistency rule that enforces the constraint that for each element shared by multiple contexts its instances (or values) must be identical for all the sharing contexts.

The results of the MurMur project can be used directly in the ontology field to define and use modules. The stamp meta-concept allows users to name the modules and to assert for each element of the ontology to which module(s) it belongs. There is no need to physically separate each module from the other modules. Specific kinds of relationships (inter-representation, multi-association) allow designers to explicitly relate elements that describe the same real world phenomena while belonging to different modules. Two rules maintain the consistency of the modules:

- Each module must be a classic database.
- Any two schema elements that belong to different modules and that share a common (non context-dependent) description must have, at the instance level, the same instances (or values).

The extension of the data manipulation and query languages enable the languages to manipulate and query the data of either a module or a set of modules.

The MurMur solutions provide means to address modularization in different ways:

- Defining a multi-contextual ontology à la MADS, made up of a set of interrelated modules. Each module being a classic ontology.
- Merging a set of already existing ontologies into a set of interrelated modules, and thus creating a multi-contextual ontology as above. The MADS model supports a merge that keeps the origins and differences of the representations.

- Partitioning an already existing classic ontology into a set of modules by stamping each of its elements (concept, role, is-a link, attribute, and instance) with the name of the module(s) to which it will belong.

Chapter 14

Distributed and Modular Ontology Reasoning

by LUCIANO SERAFINI, ANDREI TAMILIN

As it is pointed out in the introduction to the deliverable, the question of ontology modularization can be perceived from two polar perspectives.

(1) On the one hand, the modularization can be seen as the process that decomposes, partitions, a large ontology into a set of smaller ones, modules. This approach is addressed in the preceding chapters.

(2) On the other hand, the semantic web can be rationally assumed to contain multiple distributed ontologies, modules, and the modularization therefore can be seen as a mechanism for assembling some of these modules into a coherent network that can be referred to as a single entity, modular ontology. In this chapter we pursue the later option standing for a *compositional approach* to the modularization. Nevertheless, the theoretical and practical results to be presented further can be also used for dealing with a decomposed, modularized, ontology.

Hereinafter we refer to an ontology as *modular* when it is composed from a set of autonomous ontological modules, which are interrelated between each other through semantically meaningful links (see deliverables of the KnowledgeWeb Workpackage 2.2 for more information about what the semantic links between ontologies are [BEE⁺05]).

Our approach to modules composition is formally grounded on the theory of Distributed Description Logics (DDLs) [BS03]. According to DDLs, a modular ontology formally corresponds to a set of local T-boxes (one for each ontological module) which are interrelated by sets of “bridge rules” (semantic links between modules). In this chapter we give an overview of basic definitions, properties and mechanisms of DDLs.

In particular:

- we introduce an approach which views the bridge rules connecting two ontologies

as describing an *operator* that propagates knowledge in the form of Description Logics subsumption axioms. This is used as the basis of a characterization of distributed DL reasoning using a fixed point operator, which does forward-propagation of axioms;

- we give a sound and complete *distributed tableaux* algorithm that determines the satisfiability of a *SHIQ* [HST99] concept in the context of the local axioms of an ontology and the extra knowledge imparted by the bridge rules;
- we describe the design and implementation principles of a distributed *reasoning system*, called DRAGO (Distributed Reasoning Architecture for a Galaxy of Ontologies)¹, that implements the distributed tableaux algorithm for the case when ontologies are expressed in OWL [BvHH⁺04] and interrelated by semantic links in C-OWL [BGvH⁺03, BGv⁺04];
- we conclude with some *preliminary evaluation* of the scaling behaviour of the introduced distributed reasoning approach and highlight the future directions to be taken.

14.1 Distributed Description Logics

We briefly recall the definition of DDL as given by Borgida and Serafini [BS03].

Syntax Given a non-empty set I of indexes, used to enumerate local ontologies, let $\{\mathcal{DL}_i\}_{i \in I}$ be a collection of description logics.² For each $i \in I$, let us denote a T-box of \mathcal{DL}_i as \mathcal{T}_i .³ To make every description D distinct, we will prefix it with the index of ontology it belongs to, as in $i : C$. We use $i : C \sqsubseteq D$ to say that $C \sqsubseteq D$ is being considered in the i -th ontology.

Semantic mappings between different ontologies are expressed via *bridge rules*. A bridge rule from i to j is an expression, which in the deliverable is restricted to being one of the following two forms:

$$\begin{aligned} i : x \xrightarrow{\sqsubseteq} j : y & \quad \text{— an into-bridge rule} \\ i : x \xrightarrow{\supseteq} j : y & \quad \text{— an onto-bridge rule} \end{aligned}$$

¹<http://trinity.dit.unitn.it/drago>

²We assume the reader is familiar with description logics and related reasoning systems, as described in [BCM⁺03].

³We assume that a T-box will contain all the information necessary to define the terminology of a domain, including not just concept and role definitions, but also general axioms relating descriptions, as well as declarations such as the transitivity of certain roles. This is in keeping with the intent of the original paper introducing the terms T-box and A-box.

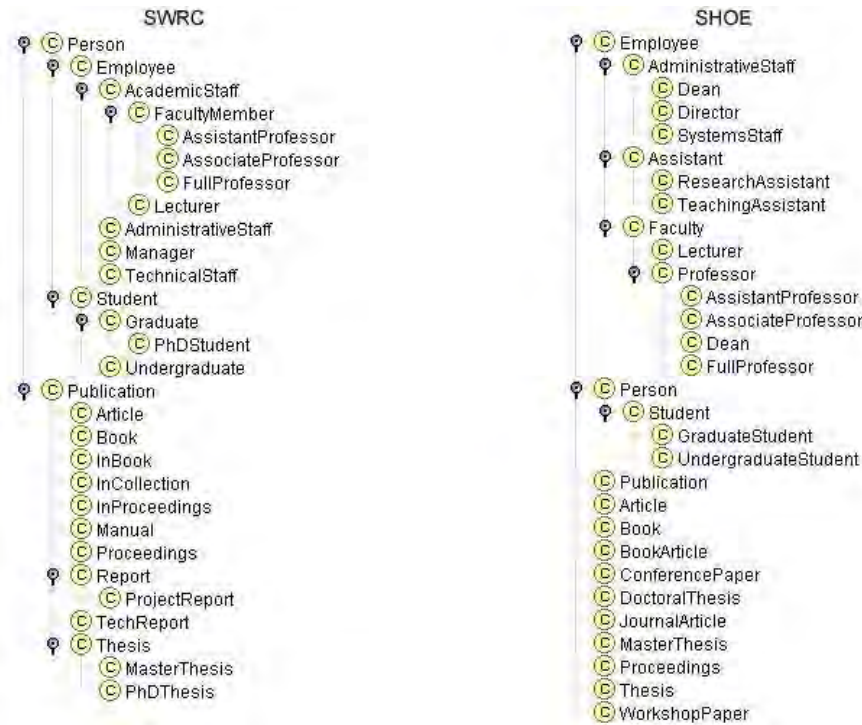


Figure 14.1: Extracts of the class hierarchies

where x and y are concepts. The derived bridge rule $i : x \stackrel{\equiv}{\mapsto} j : y$ can be defined as the conjunction of the corresponding into- and onto-bridge rule.

Bridge rules from i to j express relations between i and j viewed from the *subjective* point of view of the j -th ontology. For example, the into-bridge rule $i : C \stackrel{\sqsubseteq}{\mapsto} j : D$ intuitively says that, from the j -th point of view, the individuals in concept C in i correspond (via an approximation introduced by an implicit semantic domain relation) to a subset of the individuals in its local concept D . Therefore, bridge rules from i to j provide the possibility of translating *into* j 's ontology some of the concepts of a foreign ontology i .

A *distributed T-box (DTBox)* $\mathcal{T} = \langle \{\mathcal{T}_i\}_{i \in I}, \mathfrak{B} \rangle$ therefore consists of a collection of T-boxes $\{\mathcal{T}_i\}_{i \in I}$, and a collection of bridge rules $\mathfrak{B} = \{\mathfrak{B}_{ij}\}_{i \neq j \in I}$ between them.

Example 14.1.1 Figure 14.1 shows fragments of class hierarchies from two ontologies, SWRC⁴ and SHOE⁵, available from the DAML on-line library. These can be viewed as local T-boxes. For the sake of demonstrating the value of mappings, we considered oversimplified SHOE ontology without imports. The following are examples of bridge

⁴www.semanticweb.org/ontologies/swrc-onto-2000-09-10.daml

⁵www.cs.umd.edu/projects/plus/DAML/onts/univ1.0.daml

rules from SWRC to SHOE:

$$\text{SWRC} : \text{Publication} \xrightarrow{\equiv} \text{SHOE} : \text{Publication} \quad (14.1)$$

$$\text{SWRC} : \text{InProceedings} \xrightarrow{\sqsubseteq} \text{SHOE} : \text{ConferencePaper} \sqcup \text{WorkshopPaper} \quad (14.2)$$

$$\text{SWRC} : \text{InBook} \xrightarrow{\supseteq} \text{SHOE} : \text{BookArticle} \quad (14.3)$$

Semantics DDL semantics is a customization of the Local Models Semantics for Multi Context Systems [GG01, GS00]. Each ontology \mathcal{T}_i is *locally interpreted* by a standard DL interpretation $\mathcal{I}_i = \langle \Delta^{\mathcal{I}_i}, \cdot^{\mathcal{I}_i} \rangle$. Since local domains may be heterogeneous (e.g., time may be represented by Rationals and Integers in two ontologies), we need relations that model semantic correspondences between heterogeneous domains. A *domain relation* r_{ij} from $\Delta^{\mathcal{I}_i}$ to $\Delta^{\mathcal{I}_j}$ is a subset of $\Delta^{\mathcal{I}_i} \times \Delta^{\mathcal{I}_j}$. For example, if $\Delta^{\mathcal{I}_1}$ and $\Delta^{\mathcal{I}_2}$ are the Rationals and the Naturals, then r_{12} could be the round-off function. We use $r_{ij}(d)$ to denote $\{d' \in \Delta^{\mathcal{I}_j} \mid \langle d, d' \rangle \in r_{ij}\}$; for $D \subseteq \Delta^{\mathcal{I}_i}$, we use $r_{ij}(D)$ for $\bigcup_{d \in D} r_{ij}(d)$.

A *distributed interpretation* $\mathfrak{J} = \langle \{\mathcal{I}_i\}_{i \in I}, \{r_{ij}\}_{i \neq j \in I} \rangle$ of a DTBox \mathfrak{T} therefore combines the above two notions and is said to satisfy (written $\mathfrak{J} \models_d \mathfrak{T}$) the elements of \mathfrak{T} if

1. $\mathcal{I}_i \models_d A \sqsubseteq B$ for all $A \sqsubseteq B$ in \mathcal{T}_i
2. $\mathfrak{J} \models_d i : x \xrightarrow{\sqsubseteq} j : y$, if $r_{ij}(x^{\mathcal{I}_i}) \subseteq y^{\mathcal{I}_j}$
3. $\mathfrak{J} \models_d i : x \xrightarrow{\supseteq} j : y$, if $r_{ij}(x^{\mathcal{I}_i}) \supseteq y^{\mathcal{I}_j}$
4. $\mathfrak{J} \models_d \mathfrak{T}$, if for every $i, j \in I$, $\mathfrak{J} \models_d \mathcal{T}_i$ and $\mathfrak{J} \models_d \mathfrak{B}_{ij}$

Finally, $\mathfrak{T} \models_d i : C \sqsubseteq D$ (read as “ \mathfrak{T} d-entails $i : C \sqsubseteq D$ ”) if for every \mathfrak{J} , $\mathfrak{J} \models_d \mathfrak{T}$ implies $\mathfrak{J} \models_d i : C \sqsubseteq D$. We say \mathfrak{T} is *satisfiable* if there exists a \mathfrak{J} such that $\mathfrak{J} \models_d \mathfrak{T}$. Concept $i : C$ is *satisfiable* with respect to \mathfrak{T} if there is a \mathfrak{J} such that $\mathfrak{J} \models_d \mathfrak{T}$ and $C^{\mathcal{I}_i} \neq \emptyset$.

On injective domain correspondences A key novelty of the semantic mappings in DDL is support for multiple levels of granularity and perspective: allowing individuals to be related using arbitrary binary relations rather than just bijection. For example, while it is traditional to state correspondences such as “Wife in Ontology 1 corresponds to Moglie in Ontology 2”, DDLs support domain correspondences that are general binary relations, so that one can say that “Husband and Wife in ontology 1 correspond to Couple in Ontology 2”, which can be formalized by using onto-bridge rules $\{1 : \text{Wife} \xrightarrow{\supseteq} 2 : \text{Couple}, 1 : \text{Husband} \xrightarrow{\supseteq} 2 : \text{Couple}\}$. In [BCG04], DDLs are faulted because the collection of bridge rules $\{1 : \text{Bird} \xrightarrow{\supseteq} 2 : \text{Penguin}, 1 : \neg \text{Fly} \xrightarrow{\supseteq} 2 : \text{Penguin}\}$ do not render Penguin unsatisfiable even if Bird is subsumed by Fly in Ontology 1. As the example involving Couple shows, the general formal pattern is correct in some cases, so this is actually a problem of incomplete modeling.

In the case of Penguins, the extra information is that the domain relation is one-to-one. In such cases, one should also have added bridge rules stating that *non*-birds and flying

objects are *non-penguins*: $\{1 : \neg\text{Bird} \xrightarrow{\sqsubseteq} 2 : \neg\text{Penguin}, 1 : \text{Fly} \xrightarrow{\sqsubseteq} 2 : \neg\text{Penguin}\}$. This would indeed lead to the conclusion $\mathfrak{T} \models_d 2 : \text{Penguin} \sqsubseteq \perp$.

Since the property that the domain relation is one-one over some concept B arises quite frequently, we might consider adding a new kind of bridge rule to express it, writing something like $\xrightarrow{\leq 1} \text{Penguin}$. Interestingly, it can be proven that in the context of DDLs, such rules can be eliminated by syntactically manipulating the DTBox, so that whenever $\xrightarrow{\leq 1} G$ and $A \xrightarrow{\sqsupseteq} H$ are present, a new bridge rule $\neg A \xrightarrow{\sqsubseteq} \neg(H \sqcap G)$ is added. The tableaux technique in Section 14.4 could however use such rules more efficiently without the encoding.

Properties and Desiderata for DDL We first give some basic ways in which subsumption and a combination of onto- and into-bridge rules allows to propagate subsumptions across ontologies.

Lemma 14.1.1 *If \mathfrak{B}_{ij} contains $i : A \xrightarrow{\sqsupseteq} j : G$ and $i : B \xrightarrow{\sqsubseteq} j : H$, then $\mathfrak{T} \models_d i : A \sqsubseteq B \implies \mathfrak{T} \models_d j : G \sqsubseteq H$*

Thus, in Example 14.1.1, the subsumption $SHOE : \text{BookArticle} \sqsubseteq \text{Publication}$ can be inferred in DDL through bridge rules (14.1) and (14.3), and the subsumption $\text{InBook} \sqsubseteq \text{Publication}$ contained in \mathcal{T}_{SWRC} .

If the local languages support disjunction as a concept constructor then a more general form of propagation can occur:

Lemma 14.1.2 *If \mathfrak{B}_{ij} contains $i : A \xrightarrow{\sqsupseteq} j : G$ and $i : B_k \xrightarrow{\sqsubseteq} j : H_k$ for $1 \leq k \leq n$ (with $n \geq 0$), then $\mathfrak{T} \models_d i : A \sqsubseteq \bigsqcup_{k=1}^n B_k \implies \mathfrak{T} \models_d j : G \sqsubseteq \bigsqcup_{k=1}^n H_k$*

Additional properties would be desirable for DDL entailment. In particular, since the intended meaning is that bridge rules \mathfrak{B}_{ij} constitute a semantic channel which allows for ontology j to access and import knowledge *from* ontology i , we want information flow to be “directional” in some sense. To express this, we first introduce the notion of a bridge graph.

Definition 14.1.1 *The bridge graph $G_{\mathfrak{T}}$ of a DTBox \mathfrak{T} is a directed graph with an arc from i to j exactly when the set of bridge rules \mathfrak{B}_{ij} is non-empty.*

We can then state the main property we are looking for as:

Directionality desideratum *If in $G_{\mathfrak{T}}$ there is no path from i to j , then $\mathfrak{T} \models_d j : A \sqsubseteq B$ if and only if $\mathfrak{T}' \models_d j : A \sqsubseteq B$, where \mathfrak{T}' is obtained by removing \mathcal{T}_i , \mathfrak{B}_{ki} , and \mathfrak{B}_{ik} from \mathfrak{T} .*

This says that knowledge is propagated *only* through bridge rules, so that if there are no bridge rules that go from i towards j , then j is not affected by i . The following two isolation properties are special cases of this:

Isolation 1 A T-box without incoming bridge rules is not affected by other T-boxes. (Formally, if $\mathfrak{B}_{ki} = \emptyset$ for all $k \neq i \in I$, then $\mathfrak{T} \models_d i : A \sqsubseteq B \implies \mathcal{T}_i \models A \sqsubseteq B$)

Isolation 2 A T-box without outgoing bridge rules does not affect the other T-boxes.

Unfortunately, property *Isolation 1* does not always hold, because of onto-rules. Indeed, in the presence of onto-rule $1 : A \xrightarrow{\exists} 2 : G$ from T-box \mathcal{T}_1 to \mathcal{T}_2 , if \mathcal{T}_2 entails $\top \sqsubseteq G$, then $1 : A$ cannot be empty according to DDL semantics, and so, for example, an inconsistency would be generated if \mathcal{T}_1 entails $A \sqsubseteq \perp$. This is despite the fact that the bridge rules are toward \mathcal{T}_2 .

Property *Isolation 2* may also not hold. Indeed, if \mathcal{T}_1 is unsatisfiable, then $\mathcal{T}_2 \models_d 2 : X \sqsubseteq Y$ for every X, Y , even if there are no bridge rules connecting \mathcal{T}_1 with \mathcal{T}_2 , because there are no satisfying distributed interpretations at all. Note that in a DDL, inconsistency may arise in a connected group of T-boxes even if each T-box is locally consistent; e.g., consider the case in the hypothesis of Lemma 14.1.1, when $\mathcal{T}_j \models \top \sqsubseteq G$ and $\mathcal{T}_j \models H \sqsubseteq \perp$.

This is a significant problem, because a localized inconsistency spreads and contaminates reasoning in *all* other local ontologies, even in the absence of connections to them, because there will be no satisfying distributed interpretation, and hence every statement about them is true, as usual in logic. This problem plagues all modular and distributed representation systems.

In the following section we propose an extension of the initial semantics in order to fix this problem.

14.2 Inconsistency in DDL

There are a number of possible approaches to handle the problem of inconsistency propagation.

(1) Define d-entailment in a 2-step manner, first eliminating local T-boxes that are inconsistent, and then using the standard definition. The problem with this approach is that it is non-monotonic, and it does not deal with cases where the inconsistency arises due to several connected local sources.

(2) Use some variant of a multi-modal epistemic semantics, which allows for models of even inconsistent knowledge in the case when the set of accessible worlds is empty. Such an approach was used in [GS00] for Distributed First Order Logics, but its computational complexity/decidability aspects are quite worrisome, and the precise impact of

such non-standard semantics on logical consequences is hard to explain in an intuitive manner to users.

(3) Introduce some special interpretation, called a “hole” in [BS03], whose role is to interpret even inconsistent local T-boxes. We pursue this latter option.

Definition 14.2.1 *A hole for a T-box \mathcal{T} is an interpretation $\mathcal{I}^\epsilon = \langle \emptyset, \cdot^\epsilon \rangle$, where the domain is empty.*

Of course, the important property of holes is that $\mathcal{I}^\epsilon \models X \sqsubseteq Y$ for every X and Y , since both sides are interpreted as the empty set. We will however continue to refer to T-boxes as “inconsistent/unsatisfiable” in case there are no interpretations *other* than \mathcal{I}^ϵ which satisfy all the axioms in it.

Let us extend the notion of d-entailment \models_d , obtaining the \models_ϵ relation, by also allowing holes as interpretations for local T-boxes. Note that now even if some local T-box \mathcal{T}_i is inconsistent, we still have an interpretation for the whole DTBox: one that uses \mathcal{I}^ϵ to satisfy \mathcal{T}_i .

Properties of the semantics with holes First, the new semantics does the intended job:

Theorem 14.2.1 *The earlier-stated “directionality desideratum” holds for \models_ϵ .*

Non-standard semantics (such as multivalued logics) can however distort the meaning of the original semantics in unpredictable ways. The following results should be reassuring in this respect.

For any \mathfrak{T} and any $i \in I$, let $\mathfrak{T}(\epsilon_i)$ (the distributed T-box with the i -th local T-box viewed as inconsistent) be obtained by removing \mathcal{T}_i , \mathfrak{B}_{ij} and \mathfrak{B}_{ji} from \mathfrak{T} , and by extending each \mathcal{T}_j with the set of axioms $\{G \sqsubseteq \perp \mid i : A \xrightarrow{\exists} j : G \in \mathfrak{B}_{ij}\}$. For any finite set $J = \{i_1, \dots, i_n\}$, such that $J \subset I$, let $\mathfrak{T}(\epsilon_J)$ be $\mathfrak{T}(\epsilon_{i_1}) \dots (\epsilon_{i_n})$. (If J is empty $\mathfrak{T}(\epsilon_J) = \mathfrak{T}$.) The following then precisely characterizes the relationship of \models_d and \models_ϵ :

Proposition 14.2.1 *$\mathfrak{T} \models_\epsilon i : X \sqsubseteq Y$ if and only if for every subset $J \subseteq I$ not containing i , $\mathfrak{T}(\epsilon_J) \models_d i : X \sqsubseteq Y$.*

Moreover, in acyclic cases the relationship is even clearer:

Proposition 14.2.2 *Let $\mathfrak{T} = \langle \mathcal{T}_1, \mathcal{T}_2, \mathfrak{B}_{12} \rangle$ be a DTBox. Then*

- (i) *if \mathcal{T}_1 is consistent, then for $j \in \{1, 2\}$, $\mathfrak{T} \models_\epsilon j : X \sqsubseteq Y$ if and only if $\mathfrak{T} \models_d j : X \sqsubseteq Y$.*
- (ii) *if \mathcal{T}_1 is inconsistent, then $\mathfrak{T} \models_\epsilon 2 : X \sqsubseteq Y$ if and only if $\mathcal{T}_2 \cup \{G \sqsubseteq \perp \mid 1 : A \xrightarrow{\exists} 2 : G \in \mathfrak{B}_{12}\} \models X \sqsubseteq Y$*

Application to Other Frameworks As noted earlier, the problem of local inconsistency polluting the inferences of all the modules in a modular representation is quite general. We examine how the approach presented here can be applied to two previously proposed schemes.

[SK03a] proposes an elegant notion of modular ontology which starts from the semantic framework of DDLs, but restricts bridge rules to “identities” defining new local names $j : N$ using concepts $i : C$ from T-box i , modulo a semantic domain correspondence exactly like r_{ij} for DDLs.⁶ This can be modeled by replacing every definition $i : C \equiv j : N$ by the composed bridge rule $i : C \xrightarrow{\equiv} j : N$. Therefore the semantics involving holes introduced in the previous section can be applied to this approach, in order to localize inconsistencies in modules.

The Somewhere Peer Data Management System [Rou04] consists of a collection of peers which maintain local ontologies, including repositories of “extensional data”. Peers are acquainted with neighbours, whose concepts they can use, and query processing involves intensional distributed reasoning for query rewriting. Since this reasoning is (semantically) based on a single global interpretation, it is subject to the above mentioned difficulties due to inconsistency. In fact, for completeness, a *global* consistency check, involving even unconnected peers, would be required. We would suggest adopting a distributed semantics with holes, such as that of DDL. In particular, current peer links in Somewhere can be reduced to subsumption expressions like $1 : C \sqcap 3 : D \sqsubseteq 2 : E$. A DDL can be constructed from this by replacing occurrences of $i : C$ in peer j by new, local symbol C_i , and adding bridge rules $i : C \xrightarrow{\equiv} j : C_i$ and $i : \neg C \xrightarrow{\equiv} j : \neg C_i$. Our semantics then provides for directionality and locality, and the next section provides a distributed satisfiability testing algorithm for the semantics with holes.

Finally, the C-OWL [BGv⁺04] proposal for contextualized ontologies, uses a similar model theory as DDL. The only difference concerns the definition of hole, which, in C-OWL was defined on a non empty set. Those notion of hole supports directionality in all the cases except the case of presence of $i : \perp \xrightarrow{\equiv} j : \perp$ rule, which allows to propagate back inconsistency. The particular version of “holes” given in the deliverable gives to C-OWL the directionality property, in addition to the localized inconsistency it already had.

14.3 Fixed-Point Semantics of Bridge Rules

As we saw earlier, combinations of bridge rules allow the propagation of subsumptions across T-boxes. To better understand how this propagation happens, we will associate with a set \mathcal{B}_{ij} of bridge rules an operator of the same name, which extends the j -th T-box with

⁶Although [SK03a] originally defines imported names using conjunctive queries over concepts and roles in T-box j , it then says that these can be “rolled up” into descriptions. Although this may in fact not always be doable, we will deal here with exactly those definitions for which this roll-up holds.

a set of subsumption axioms that are the transformation via bridge rules of subsumptions in the i -th T-box.

Before proceeding further, we need to introduce the concept of disjoint union for interpretations. To begin with, we define as usual the disjoint union of two, possibly overlapping sets S and T as $S \uplus T = (S \times \{\#\}) \cup (T \times \{\@\})$, where the values are distinguished by tupling with two discriminant symbols — $\#$ and $\@$, in this case. This is generalized to the disjoint union $\biguplus_{i \in K} S_i$ of a collection of sets $\{S_i\}_{i \in K}$ indexed with (possibly infinite) K , by using the indexes i as the discriminants.

Definition 14.3.1 *Given two interpretations $\mathcal{I} = \langle \Delta_{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ and $\mathcal{J} = \langle \Delta_{\mathcal{J}}, \cdot^{\mathcal{J}} \rangle$ of the same language \mathcal{L} , the disjoint union of \mathcal{I} and \mathcal{J} , denoted by $\mathcal{I} \uplus \mathcal{J}$, is $\langle \Delta_{\mathcal{I} \uplus \mathcal{J}}, \cdot^{\mathcal{I} \uplus \mathcal{J}} \rangle$, where:*

1. $\Delta_{\mathcal{I} \uplus \mathcal{J}} = \Delta_{\mathcal{I}} \times \{\#\} \cup \Delta_{\mathcal{J}} \times \{\@\}$
2. for concept A , $(A)^{\mathcal{I} \uplus \mathcal{J}} = A^{\mathcal{I}} \times \{\#\} \cup A^{\mathcal{J}} \times \{\@\}$
3. for role R , $R^{\mathcal{I} \uplus \mathcal{J}} = \{ \langle (x, \#), (y, \#) \rangle \mid \langle x, y \rangle \in R^{\mathcal{I}} \} \cup \{ \langle (w, \@), (z, \@) \rangle \mid \langle w, z \rangle \in R^{\mathcal{J}} \}$

Disjoint union for interpretations $\biguplus_{k \in K} \mathcal{I}_k$ can similarly be generalized to the case of a sets. Intuitively the interpretation $\mathcal{I} \uplus \mathcal{J}$ is interpretation that is composed of two unrelated subparts one is isomorphic to \mathcal{I} and the other to \mathcal{J} .

Definition 14.3.2 *A description logic family \mathcal{DL} has the disjoint union satisfiability property if $E^{\mathcal{I}'} = \biguplus_{k \in K} E^{\mathcal{I}_k}$ holds for all concepts and roles E over \mathcal{DL} , and for all interpretations $\mathcal{I}' = \biguplus_{k \in K} \mathcal{I}_k$.*

Lemma 14.3.1 *\mathcal{SHIQ} , and its sub-languages, have the distributed union satisfiability property.*

On the other hand, languages that support nominals (such as OWL), or A-boxes do not have this property.

The bridge operator The bridge operator essentially applies generalized subsumption propagation Lemma 14.1.2, to find new subsumptions:

Definition 14.3.3 *Given a set of bridge rules \mathfrak{B}_{12} from \mathcal{DL}_1 to \mathcal{DL}_2 , the bridge operator $\mathfrak{B}_{12}(\cdot)$, taking as input a T-box in \mathcal{DL}_1 and producing a T-box in \mathcal{DL}_2 , is defined as follows:*

$$\mathfrak{B}_{12}(\mathcal{T}_1) = \left\{ G \sqsubseteq \biguplus_{k=1}^n H_k \left[\begin{array}{l} \mathcal{T}_1 \models A \sqsubseteq \biguplus_{k=1}^n B_k, \\ 1 : A \xrightarrow{\exists} 2 : G \in \mathfrak{B}_{12}, \\ 1 : B_k \xrightarrow{\sqsubseteq} 2 : H_k \in \mathfrak{B}_{12}, \\ \text{for } 1 \leq k \leq n, n \geq 0 \end{array} \right. \right\}$$

(Notationally, $\bigsqcup_{k=1}^0 D_k$ denotes \perp .)

It is remarkable that these are essentially *all* the inferences that one can get, if we use the semantics with holes:

Theorem 14.3.1 *Let $\mathfrak{T}_{12} = \langle \mathcal{T}_1, \mathcal{T}_2, \mathfrak{B}_{12} \rangle$ be a distributed T-box. If \mathcal{DL}_1 and \mathcal{DL}_2 have the distributed union satisfiability property then:*

$$\mathfrak{T}_{12} \models_{\epsilon} 2 : X \sqsubseteq Y \iff \mathcal{T}_2 \cup \mathfrak{B}_{12}(\mathcal{T}_1) \models X \sqsubseteq Y$$

For any family $\mathfrak{B} = \{\mathfrak{B}_{ij}\}_{i,j \in I}$ of bridge rules, we can combine these into a new operator \mathfrak{B} on a family of T-boxes as follows:

$$\mathfrak{B}(\{\mathcal{T}_i\}_{i \in I}) = \left\{ \mathcal{T}_i \cup \bigcup_{j \neq i} \mathfrak{B}_{ji}(\mathcal{T}_j) \right\}_{i \in I}$$

If I is finite and each \mathfrak{B}_{ij} is finite, then there is a positive integer b such that for every family of T-boxes \mathbf{T} , $\mathfrak{B}^b(\mathbf{T}) = \mathfrak{B}^{b+1}(\mathbf{T})$. Let us then define $\mathfrak{B}^*(\mathbf{T})$ as $\mathfrak{B}^b(\mathbf{T})$, where b is the first positive integer such that $\mathfrak{B}^b(\mathbf{T}) = \mathfrak{B}^{b+1}(\mathbf{T})$. Furthermore let $\mathfrak{B}^{b+1}(\mathbf{T})_i$, be the i -th T-box in $\mathfrak{B}^{b+1}(\mathbf{T})$.

Theorem 14.3.2 *For every $\mathfrak{T} = \langle \mathbf{T}, \mathfrak{B} \rangle$, $\mathfrak{T} \models_{\epsilon} j : X \sqsubseteq Y$ if and only if the j -th T-box of $\mathfrak{B}^*(\mathbf{T})$ entails $X \sqsubseteq Y$.*

Applications to Caching A number of researchers have considered the idea of caching locally the necessary information from the imported ontology \mathcal{T}_{other} , since this is assumed to be both more efficient (there is no need to interrupt local reasoning, while waiting for answers from the other ontology), and more perspicuous from the point of view of the local user: in order to understand an imported concept F , it is not necessary to understand *all* of \mathcal{T}_{other} , only the locally cached part, which is presumed to be much smaller. (This idea is also known as “subsetting”, and there is considerable research on this topic in the ontology community .)

Theorem 14.3.2 above indicates that it is possible to finitely pre-compile in a sound and complete manner the subsumption information imported into a local ontology \mathcal{T}_j by the bridge rules in a DTBox \mathfrak{T} : compute and store it.

In a similar vein, [SK03a] takes the set of imported concept definitions $\{N_k \equiv other : D_k \mid k = 1, \dots, n\}$, and then computes and caches the subsumption hierarchy of the $\{N_k\}$. Since we have explained in Section 14.2 that the module mechanism in [SK03a] can be represented as a DDL, Lemma 14.1.2 indicates that if the language contains at least \mathcal{ALC} , and if it is possible to ask subsumption queries about complex concepts composed using the imported definitions, then it is not sufficient to cache only subsumptions of the form $D_{k1} \sqsubseteq D_{k2}$, since there may be additional subsumptions entailed, involving

disjunctions. On the other hand, by Theorem 14.3.2 it is *sufficient* to cache all subsumptions of the form $N_k \sqsubseteq N_{k_1} \sqcup \dots \sqcup N_{k_m}$, whose definitions satisfy the condition⁷ $\mathcal{T}_{other} \models D_k \sqsubseteq D_{k_1} \sqcup \dots \sqcup D_{k_m}$.

14.4 Distributed Tableaux Algorithm for DDL

In this section we describe a tableaux-based decision procedure for $\mathfrak{T} \models_{\epsilon} i : X \sqsubseteq Y$, for DTBoxes whose bridge graph $G_{\mathfrak{T}}$ is acyclic. The cyclic case is left for future work, pending the identification of a loop blocking strategy that preserves the independence of the local proofs.

To simplify the description, we suppose that local ontologies are expressed in (a subset of) the \mathcal{SHIQ} language — one of the most widely known DLs. Also, we will assume that the consequences of bridge rules are atomic names. (This condition can easily be achieved by introducing, through definitions, names for the consequent concepts.). We need the usual notion of axiom internalization, as in [HST99]: given a T-box \mathcal{T}_i , the concept $C_{\mathcal{T}_i}$ is defined as $C_{\mathcal{T}_i} = \prod_{E \sqsubseteq D \in \mathcal{T}_i} \neg E \sqcup D$; also, the role hierarchy $R_{\mathcal{T}_i}$ contains the role axioms of \mathcal{T}_i , plus additional axioms $P \sqsubseteq U$, for each role P of \mathcal{T}_i , with U some fresh role.

The algorithm for testing j -satisfiability of a concept expression X (i.e., checking $\mathfrak{T} \not\models_{\epsilon} j : X \sqsubseteq \perp$) builds, as usual, a finite representation of a distributed interpretation \mathfrak{I} , by running local *autonomous* \mathcal{SHIQ} tableaux procedures to find each local interpretation \mathcal{I}_i of \mathfrak{I} .

Definition 14.4.1 *For each $j \in I$, the function \mathbf{DTab}_j takes as input a concept X and tries to build a representation of \mathcal{I}_j with $X^{\mathcal{I}_j} \neq \emptyset$ (called a completion tree [HST99]) for the concept $X \sqcap C_{\mathcal{T}_j} \sqcap \forall U.C_{\mathcal{T}_j}$, using the \mathcal{SHIQ} expansion rules, w.r.t. the role hierarchy $R_{\mathcal{T}_j}$, plus the following additional “bridge” expansion rules*

⁷There is no need to iterate if we assume that imported names cannot be used in additional axioms of the local ontology — only for labeling information on the semantic web, for example.

Unsat- \mathfrak{B}_{ij} -rule

if 1. $G \in \mathcal{L}(x)$, $i : A \xrightarrow{\exists} j : G \in \mathfrak{B}_{ij}$, and
 2. $IsSat_i(A \sqcap \neg \sqcup \mathbf{B}') = False$, for some $\mathbf{H}' \not\subseteq \mathcal{L}(x)$,
 then $\mathcal{L}(x) \longrightarrow \mathcal{L}(x) \cup \{\sqcup \mathbf{H}'\}$

New- \mathfrak{B}_{ij} -rule

if 1. $G \in \mathcal{L}(x)$, $i : A \xrightarrow{\exists} j : G \in \mathfrak{B}_{ij}$, and
 2. $\mathbf{B} \subseteq \{B \mid i : B \xrightarrow{\exists} j : H \in \mathfrak{B}_{ij}\}$, and
 3. for no $\mathbf{B}' \subseteq \mathbf{B}$ is $IsSat_i(A \sqcap \neg \sqcup \mathbf{B}') = False$, and
 4. for no $\mathbf{B}' \supseteq \mathbf{B}$ is $IsSat_i(A \sqcap \neg \sqcup \mathbf{B}') = True$,
 then if $\mathbf{DTab}_i(A \sqcap \neg \sqcup \mathbf{B}) = Satisfiable$
 then $IsSat_i(A \sqcap \neg \sqcup \mathbf{B}) = True$
 else $IsSat_i(A \sqcap \neg \sqcup \mathbf{B}) = False$

The idea, inspired by bridge operator $\mathfrak{B}_{ij}(\cdot)$, is that whenever \mathbf{DTab}_j encounters a node x that contains a label G which is a consequence of an onto-bridge rule, then if $G \sqsubseteq \sqcup \mathbf{H}$ is entailed by the bridge rules, the label $\sqcup \mathbf{H}$, is added to x . To determine if $G \sqsubseteq \sqcup \mathbf{H}$ is entailed by bridge rules \mathfrak{B}_{ij} , \mathbf{DTab}_j invokes \mathbf{DTab}_i on the satisfiability of the concept $A \sqcap \neg(\sqcup \mathbf{B})$. \mathbf{DTab}_i will build (independently from \mathbf{DTab}_j) an interpretation \mathcal{I}_i , as illustrated in Figure 14.2. To avoid redundant calls, \mathbf{DTab}_j caches the calls to \mathbf{DTab}_i in a data structure $IsSat_i$, which caches the subsumption propagations that have been computed so far. Specifically, for every C , $IsSat_i(C)$ will be set to *True/False* whenever $\mathfrak{T} \not\models_{\epsilon} i : C \sqsubseteq \perp$ is determined.

Theorem 14.4.1 (Termination) *For any acyclic DTBox \mathfrak{T} and for any SHIQ concept X , $\mathbf{DTab}_j(X)$ terminates.*

Theorem 14.4.2 (Soundness and completeness) *$j : X$ is satisfiable in \mathfrak{T} if and only if $\mathbf{DTab}_j(X)$ can generate a complete and clash-free completion tree.*

Note that the construction of the distributed interpretation can be parallelized, as each local tableau procedure can run independently from the others, without checking for blocking conditions with nodes generated by the other local tableaux. We will overview the implementation of the distributed algorithm proposed above in the next section.

14.5 DRAGO Reasoning System

In this section we will describe a design and implementation principles that lay in the base of DRAGO (Distributed Reasoning Architecture for a Galaxy of Ontologies), the system

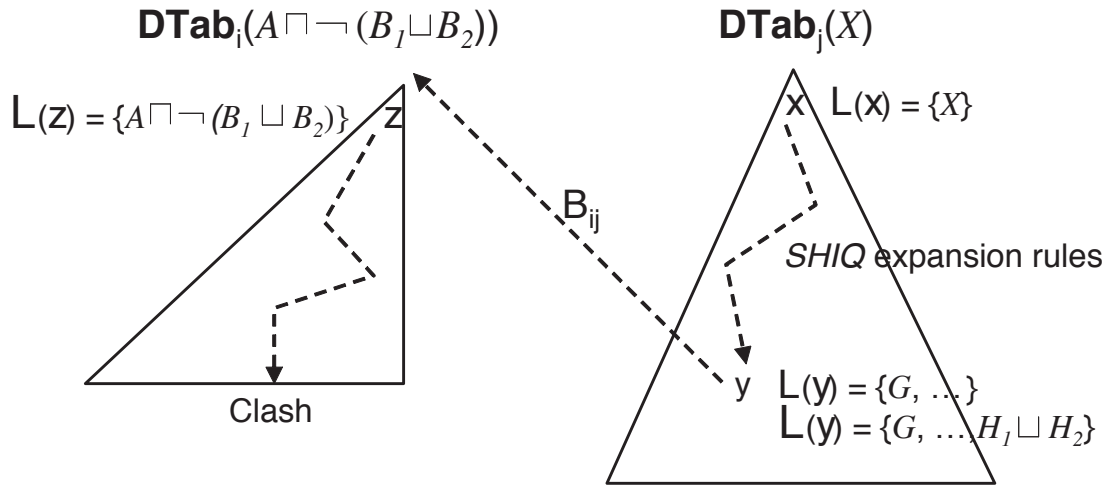


Figure 14.2: Illustrative step of the distributed tableaux algorithm: subsumption propagation forced by bridge rules $i : A \xrightarrow{\exists} j : G$, $i : B_1 \xrightarrow{\sqsubseteq} j : H_1$ and $i : B_2 \xrightarrow{\sqsubseteq} j : H_2$

for reasoning with multiple ontologies connected by pairwise semantic mappings.⁸

Vision As depicted in Figure 14.3, DRAGO envisages a Web of ontologies being distributed amongst a peer-to-peer network of *DRAGO Reasoning Peers* (DRP).

The role of a DRP is to provide reasoning services for ontologies registered to it, as well as to request reasoning services of other DRPs when this is required for fulfillment of distributed reasoning algorithm. The key issue of the DRP is that it provides possibility to register not just a stand alone ontology, but an ontology coupled with a set of semantic mappings.

In order to register an ontology to a DRP, the users specify a logical identifier for it, a Unified Resource Identifier (URI), and give a physical location of ontology on the Web, a Unified Resource Locator (URL). Besides that, it is possible to assign to an ontology a set of semantic mappings, providing in the same manner their location on the Web. As we discussed in the previous sections, attaching mappings to ontology enriches its knowledge due to the subsumption propagation mechanism. To prevent the possibility of attaching malicious mappings that can obstruct or falsify reasoning services, only the user that registered the ontology is allowed to add mappings to it.

When users or applications want to perform reasoning with a one of registered ontologies, they refer to the corresponding DRP and invoke its reasoning services giving the URI of the desired ontology.

⁸<http://trinity.dit.unitn.it/drago>

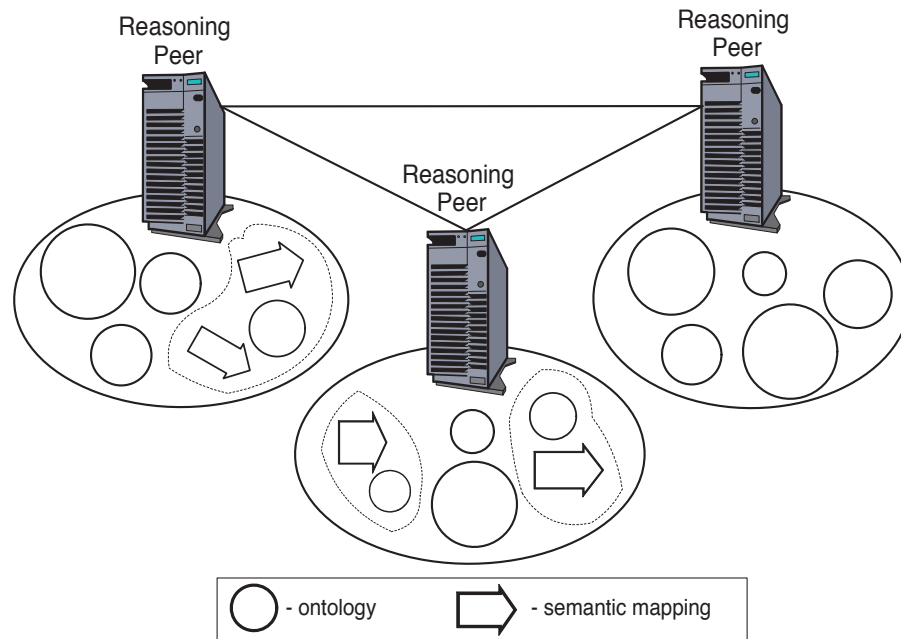


Figure 14.3: DRAGO vision.

Architecture A DRP constitutes the basic element of DRAGO. The major components of a DRP are depicted in Figure 14.4.

A DRP has two interfaces which can be invoked by users or applications:

- A *Registration Service* interface is meant for creating/modifying/deleting of registrations of ontologies and mappings assigned to them.
- A *Reasoning Services* interface enables the calls of reasoning services for registered ontologies. Among the reasoning services can be a possibility to check ontology consistency, build classification, verify concepts satisfiability and check entailment.

All accessibility information about registered ontologies and mappings is stored by a DRP in its local *Registration Storage*.

In order to register an ontology with a collection of semantic mappings attached to it (both available on the Web) a user or application invokes the Registration Service of a DRP and sends to it the following registration information:

- URI to which the ontology will be bound.
- URLs of ontology and semantic mappings attached to it, if any.
- If the semantic mappings connect this ontology with ontologies registered to external DRPs then additionally the URLs of these DRPs should be specified. This

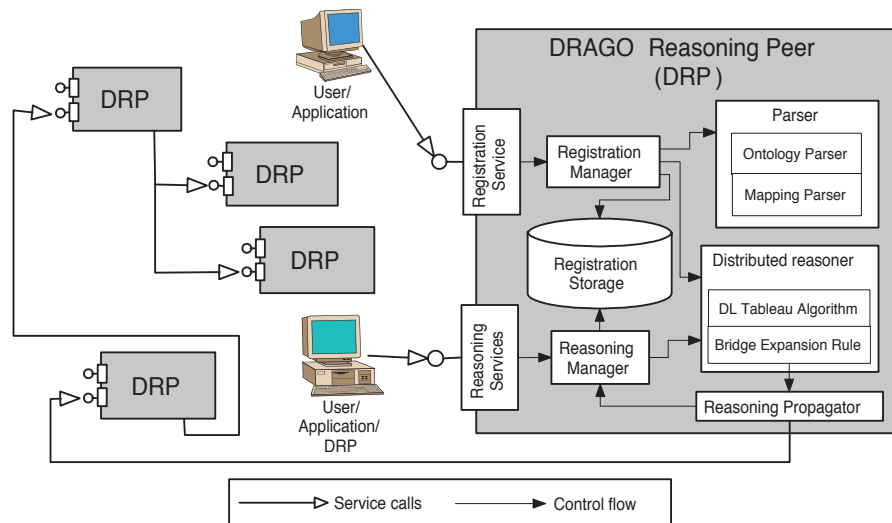


Figure 14.4: DRAGO architecture.

requirement is explained by the necessity to know who is responsible for reasoning with these ontologies.

The Registration Service interface is implemented by the *Registration Manager*. When the Manager receives a registration request, it (i) consults the Registration Storage and verifies if the URI has not occupied yet, (ii) if not it accesses ontologies and assigned mappings from their URLs, (iii) asks Parser component to process them, (iv) initializes the Distributed Reasoner with the parsed data, and (v) finally adds a new record to the Registration Storage.

The *Parser* component translates ontologies and mappings source files to the internal format used by the Distributed Reasoner. For doing so, the Parser consist from two sub components: the ontology parser, tailored on ontology language formats (for example, OWL [BvHH⁺04]), and the mapping parser, tailored on mapping formats (for example, C-OWL [BGvH⁺03]).

The *Reasoning Manager* component implements the Reasoning Services interface. When users, applications or other DRPs invoke this interface sending the URI of requested ontology, the Manager verifies with the Registration Storage whether the URI is registered to the DRP and, if yes, asks the Distributed Reasoner to execute corresponding reasoning task for that ontology.

The *Distributed Reasoner* represents a brain of a DRP. It realizes the distributed algorithm proposed in the Section 14.4 and reasons on ontologies with attached mappings that are registered to the DRP. The Distributed Reasoner is built on top of standard tableau reasoner whose algorithm was extended with the additional Bridge Expansion Rule in accordance with the distributed tableau algorithm. When the Bridge Expansion Rule is applied it analyzes semantic mappings and possibly generates reasoning sub tasks that are

required to be executed in the ontologies participating in mappings.

To dispatch the reasoning tasks generated by a Distributed Reasoner to the responsible reasoners, the *Reasoning Propagator* component refers to the Reasoning Manager and either dispatches reasoning to the local Distributed Reasoner or sends out a request of reasoning service to the corresponding external DRP.

Implementation The described DRAGO architecture has been implemented for the case of OWL [BvHH⁺04] ontology space. For expressing semantic mappings between OWL ontologies we use a C-OWL [BGvH⁺03, BGv⁺04]. According to C-OWL, mapping consists of references to the source and target ontologies and a series of bridge rules relating classes between these ontologies. Due to the limitations of introduced distributed tableau algorithm (see Section 14.4) among the possible C-OWL bridge rule types DRAGO supports the use of \equiv , \sqsubseteq , \sqsupseteq rules connecting atomic concepts.

A Distributed Reasoner was implemented as an extension to an open source OWL reasoner Pellet.⁹ Originally, Pellet parses OWL ontology to a Knowledge Base (T-box/A-box). To satisfy the needs of DRAGO we extended a Pellet's Knowledge Base with a M-box containing parsed C-OWL mappings. Another extension of Pellet was done by adding a Bridge Expansion Rule to the core tableau algorithm in order to transform it to the distributed tableau. This rule is called for every node created by the core tableau algorithm and consist in finding such bridge rules in M-box that are capable of importing new subsumptions from mapping-related ontologies. Although the proposed distributed tableaux algorithm admits cache-based implementation current version of Distributed Reasoner was implemented in a straightforward way without advanced optimization techniques as the caching, for example, is. We left optimizations for the future work and extensive testing phase.

DRAGO is implemented to operate over HTTP and to access ontologies and mappings published on the Web. A DRP represents several java servlets that should be deployed to a java-enabled Web-server, for example Tomcat.¹⁰

14.6 Preliminary Evaluation of Distributed Reasoning

In order to practically evaluate the proposed distributed reasoning algorithm we performed some preliminary experiments in order to see how distributed algorithm performs w.r.t. a global tableaux algorithm based on the encoding described in [BS03].

Experimental Methodology We used the following comparison scheme. Given a modular ontology we load it into DRAGO distributed reasoner, submit to the reasoner 100

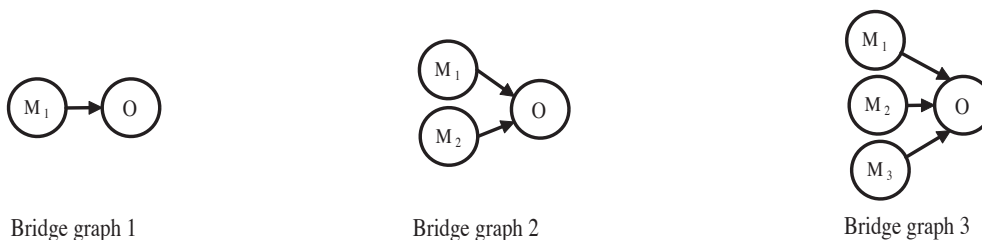
⁹<http://www.mindswap.org/2003/pellet>

¹⁰<http://jakarta.apache.org/tomcat>

random satisfiability tests, and collect the CPU time spent by the reasoner to verify tests. Then we encode the modular ontology into equivalent global ontology, load it into a Pellet OWL reasoner, submit to the reasoner the same set of 100 satisfiability tests, and again measure the CPU time spent.

As modules to be used for creating a synthetic modular ontology we have used the OWL version of the biochemistry ontology developed in Tambis project and available at on-line ontology store.¹¹ Despite the fact that the very same ontology was used as a source of multiple modules to force the difference between modules we intentionally changed the internal namespace of ontology in every module.

To interconnect modules into a modular ontology we generated random sets of bridge rules between concepts of modules (M) and target modular ontology (O). We investigated three topologies of modular ontology formally corresponding to the following bridge graphs in DDLs:



Results and Comments All tests were made on Intel Pentium M processor 1500MHz with 512MB RAM running Microsoft Windows XP Professional.

The results of the preliminary evaluation are presented on Figures 14.5, 14.6, and 14.7, where the light bar stands for invocation of distributed reasoner DRAGO and the solid bar corresponds to the stand alone Pellet OWL reasoner.

We could not proceed the experimenting with the bigger amount of bridge rules since the global approach in most of cases ran out of memory whereas the distributed approach continued to work.

14.7 Conclusions and Outlook

In this chapter we have focused on the Distributed Description Logics, the formalism supporting the representation and providing reasoning support to the case of distributed and modular ontologies.

We have described the theoretically sound and complete distributed tableau-based reasoning technique for DDLs and presented an overview of its prototypical implementation,

¹¹<http://protege.stanford.edu/plugins/owl/owl-library/tambis-full>

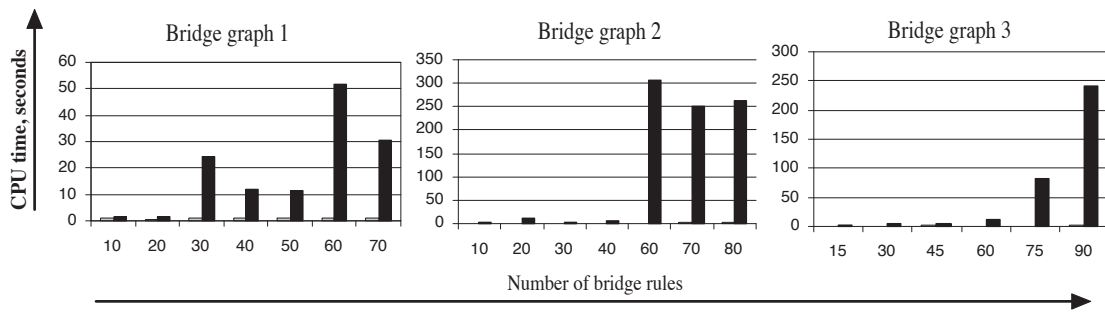


Figure 14.5: Loading ontologies into reasoners (light - distributed, solid - global).

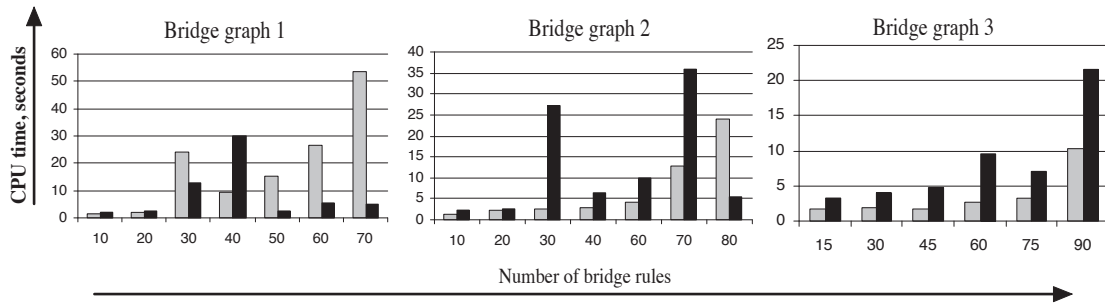


Figure 14.6: 100 random satisfiability tests (light - distributed, solid - global).

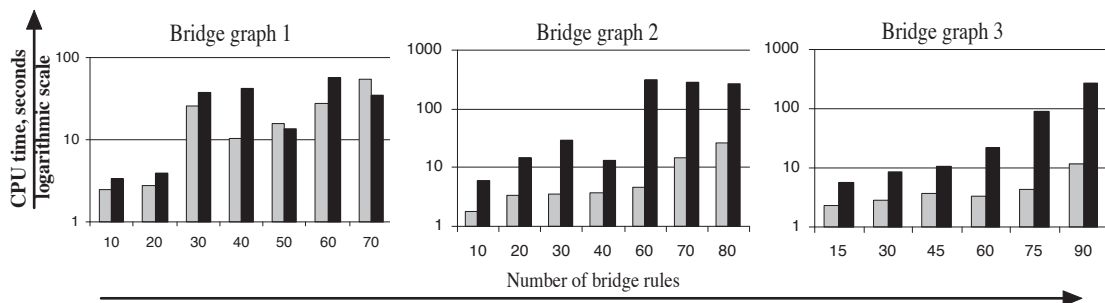


Figure 14.7: Total time spent by reasoners (light - distributed, solid - global).

DRAGO reasoning system.

Preliminary evaluation of the proposed distributed reasoning approach versus the reasoning in the equivalent global ontology forms a first demonstrative impression that the execution of reasoning tasks with a network of modules is likely to scale against the same task executed in a single ontology formed by turning modules in a network together. However, this is just a preliminary step towards investigation of scaling behaviour of the distributed algorithm.

As promising paths for further research we plan to implement and explore the caching techniques for improving the distributed algorithm, perform an extensive testing for investigation of the algorithm's scaling behaviour, and finally define and implement a cycle blocking strategy to deal with general distributed T-boxes.

Chapter 15

Reasoning on Dynamically Built Reasoning Space with Ontology Modules

by FABIO PORTO

15.1 Introduction

Reasoning over distributed and autonomously developed ontologies has to face a number of new challenges. First, current reasoners [HM03] consider ontology as forming a single logical theory. Unfortunately, both distribution and autonomy adversely contribute to such a view. Therefore in order to use current reasoning software the set of autonomously developed ontologies must be aligned and integrated into a single consistent ontology. Second, as in the context of database integration [DPS98], and to allow building a single logical theory, definition on different ontologies must be aligned by the use of correspondence expressions. Thirdly, the set of involved ontologies may get to a quite voluminous amount of data. As a result, a naïve solution of transferring all ontologies to a location and then proceeding with local reasoning does not scale up. Finally, autonomously defined ontologies may assert contradictory definitions, which some authors classify as conflicts in the integration process. Conflicts identification is, in fact, a tool for fixing correspondence assertions and applying ontology alignment. So, reasoning under this setting should be capable of identifying such conflicts and acting appropriately.

In Chapter 14, Serafini and Tamilin, present an approach for distributed reasoning, in the context discussed in the previous paragraph, based on global subsumption. The main result of their work is to deduce global subsumption based on local ontology subsumption and bridge rules [BGvH⁺03, BGv⁺04]. Their approach is scalable as it keeps reasoning to single ontologies.

In this contribution, we propose a different approach based on global reasoning over

relevant ontology entities, with respect to an ontology query, extracted from ontology modules. Ontology modules are supported by peers that provide reasoning and ontology processing. It is assumed that a set of mapping expressions exist in each module semantically associating local ontology entities to corresponding definitions in other ontology modules.

Ontology queries submitted to peers are answered by reasoning over a dynamically built reasoning space comprising relevant ontology entities captured among autonomous developed ontologies. We give some initial ideas on how to dynamically build a reasoning space and point to further research issues.

The rest of this contribution is structured as follows. Section 15.2 presents the concepts of ontology spaces and ontology modules. Next, section 15.3 develops the strategy of building a reasoning space to answer reasoning queries over an ontology space. Section 15.4 uses a scenario of web services discovery to illustrate the approach. Finally, section 15.5 gives our conclusions and points to some future work.

15.2 Ontology Space and Modules

Autonomously developed ontologies emerge quite naturally in different business areas. However as business evolves, interactions among partners promote the extension of each one's activities towards a network of interrelated process and data. If automation is required to support the business process, the independent developed ontologies may prove useful in solving semantic misunderstandings by offering independently a wider semantic cover for reasoning tasks.

We name a set of autonomously specified ontologies over which an hypothetical reasoner could evaluate an ontology query an *ontology space* (OS). Given two ontologies taking part in a OS, we say that they intersect if there is a known correspondence assertion associating entities against both ontologies.

The set of entities specified in a ontology together with a set of correspondences expressed with entities in other ontologies define an ontology module (M). The underlying ontology of a module is named its *base ontology*. An ontology entity in a module is either defined in its *base ontology*, local entity, or added to it by an equivalence correspondence with an external entity, specified in a different ontology. The concept of modules is similar to context in C-OWL [BGvH⁺03, BGv⁺04].

Definition 15.2.1 *A module is a tuple $Mo = \langle id, D, L, C, O_b, O_s \rangle$, where id corresponds to a Unique Resource Identifier (URI) for the module, D is the description of the module, either expressed in natural language or by means of an ontology language; L is the ontology language used in Mo ; C is a set of correspondences (defined below) associating local entities with entities defined in external modules; O_b is the base ontology and O_s is the set of external ontologies to which correspondences with local entities are specified.*

The ontology description should aid both humans and machines in selecting modules. Such descriptions may include domain characteristics, non-functional properties, and assumptions. The latter can be used, for instance, in deciding which modules to consider in answering a query.

Definition 15.2.2 below specifies valid correspondences between ontology entities [BGvH⁺03, BGv⁺04].

Definition 15.2.2 *An ontology correspondence is a relation in one of the following forms:*

- $C \equiv D$ (for class equivalence)
- $C \subseteq D$ (for subsumption)
- $C \supseteq D$ (for superset)
- $R \equiv S$ (for relationship equivalence)
- $v \equiv t$ (for instance equivalence)

where (C, R, v) and (D, S, t) are, respectively, local and external entities with respect to a module. C is of type class, D is a class expression of the form $f(t_1, \dots, t_n)$, where the terms t_i are either class names or class expressions and f is an n -ary class builder operator, R and S are ontology relationships, and v and t are instances [BGvH⁺03, BGv⁺04].

Correspondences are specified from a module designer point of view. They contribute to the semantic autonomy of each module by giving local interpretation to external entities, with no impact on their semantics in the original ontologies. We further consider that the ontology correspondences complements the base ontology's definitions and can be locally validated indicating eventual conflicts.

We also define a *peer* $P = \langle Mo, QL \rangle$ that models a software component capable of answering ontology queries expressed in QL language over an ontology module Mo . A *peer system* is a set $PS = \cup P_i$, where $1 \leq i \leq n$ is the identification of each peer in the set.

15.3 Reasoning Space

We use the term *reasoning space* (RS) to denote a virtual ontology that is dynamically built to answer an ontology query over an ontology space.

A reasoning space includes the base ontology associated to a module that receives the query and complementary elements gathered from external ontologies. Entities of a reasoning space share the same ontology language and form a single ontology.

Definition 15.3.1 A Reasoning space RS is defined as: $RS \subseteq \{O \cup C\}$, where O is an ontology space and C is the set of correspondences associating elements in O .

Definition 15.3.2 We also define a reasoning space mapping function $f(Q, RS, O): RS'$ that given: a ontology query Q , a reasoning space RS and a ontology space O , produces a new reasoning space RS' .

The mapping function f expands RS during query evaluation. Reasoning on a RS is done incrementally as relevant entities in external ontologies are identified and added to it. As soon as the query is decided, the incremental process terminates.

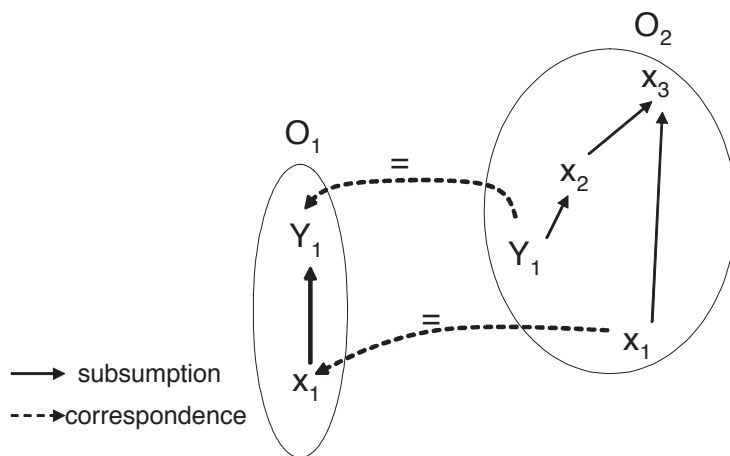


Figure 15.1: Ontology space.

Let us motivate the discussion on *reasoning space* by aid of a simple example, as illustrated in Figure 15.1. The picture presents an ontology space O , comprised of two ontologies, O_1 and O_2 . Module $M_1 = \{O_1, C_1\}$ includes its base ontology O_1 and a set of equivalence type of correspondences C_1 , associating entities defined in O_2 . One may clearly identify that the complete logical theory is inconsistent as the subsumption relation between $O_1 : y_1$ and $O_1 : x_1$ should also hold in O_2 as a result of C_1 . Unfortunately, as a result of the evolution of autonomously managed ontologies, we should expect that inconsistencies like this one are prone to emerge and should be considered when reasoning over the ontology space.

In order to complete the example, a ontology query Q , $Q = x_1 \subseteq x_2$, is submitted¹ to ontology module M_1 . Query Q can not be decided using uniquely entities specified in M_1 base ontology O_1 , therefore the mapping function $f(Q, O_1 \cup C_1, O) : RS_1$ is computed to extend the original reasoning space comprised initially of the union of ontology O_1 and the correspondence set C_1 . The mapping function f identifies a set of relevant entities

¹We consider the existence of a query answering system on top of each module forming a P2P network

in O_2 to be included into the *reasoning space* of query Q . Relevant entities are those in O_2 associated to entities in C_1 that appear in Q , $RE = \{y_1, x_2, x_3, y_1 \subseteq x_2, x_2 \subseteq x_3, x_1 \subseteq x_3\}$. The *reasoning space* RS is augmented with relevant entities in R , $RS' = O_1 \cup R$, and reasoning over RS' can proceed. Having all the entities relevant for query Q , RS' has sufficient knowledge for deciding the query. As a matter of fact, RS' will bring up the existing inconsistency in the ontology space, providing an opportunity for alignment between ontologies and correspondences.²

15.3.1 Ontology Query Model

We consider boolean DL conjunctive queries where users want to check on satisfiability with respect to a ontology space. These query types are important for applications like web service discovery, where a matching process requires to verify subsumption and equivalence between goals and web service description terms, as well as satisfiability of *instance of concept* expression [LRK04].

Our approach is based on set theory, as adopted in [BLRT05], in the context of web service discovery. In this context, a query expresses a conjunction of disjoint sets of objects.

Definition 15.3.3 *An ontology query is in reduced clause form RCF [BLRT05] if given $Q = q_1 \wedge q_2 \wedge \dots \wedge q_n$, where q_i is a clause modeling a set of objects, then $q_i \cap q_j = \emptyset$, $i \neq j$, $1 \leq i, j \leq n$.*

In our example, the query $Q = x_1 \subseteq x_2$ includes a single clause, restricting the concept x_2 .

A query in RCF is satisfied if we can prove that each of its disjoint sets is a subset of some set of objects in RS .

15.3.2 Finding Relevant Entities on the Ontology Space

As discussed above, the mapping function identifies relevant entities on the ontology space to be considered in extending the *reasoning space*. A strategy for identifying the set of relevant entities is the objective of this section.

Identifying relevant entities is achieved in two steps. In the first step, we check for relevant correspondences in the current reasoning space and, in the second step, a new query for obtaining relevant entities is submitted to the respective ontology module.

Definition 15.3.4 *A relevant correspondence defines a set of objects with a non empty intersection with a RCF query clause.*

²We do not address in this contribution solutions to conflicting situations.

As an example, for query Q and correspondences $C_1 = \{O_1: x_1 \equiv O_2: x_1, O_1: y_1 \equiv O_2: y_1\}$, we have that $x_1 \subseteq x_1$ and $x_1 \subseteq y_1$. Therefore the relevant correspondence set $RC = C_1$.

Next, we need to query the corresponding ontology modules for relevant entities. Similarly with Definition 15.3.4, the set of relevant entities in ontology modules, RE , are those concepts and roles whose corresponding object set intersects with objects in the RC set.

In our initial example, $RE = O_2 \cap RC$, thus $RE = \{y_1, x_2, x_3, y_1 \subseteq x_2, x_2 \subseteq x_3, x_1 \subseteq x_3\}$.

15.3.3 Answering Queries over the Reasoning Space

A reasoning space is obtained by successively extending a prior version. The extension includes the relevant entities obtained in the process as described in Section 15.3.2 and the correspondences fetched from the target module.

Once obtained, a traditional reasoner evaluates the query over the reasoning space. The process finishes when, either the query has been decided or there is no more possible extension of the reasoning space.

In case an inconsistency is detected a user intervention may be requested to allow for process continuation.

15.3.4 Dealing with Global Interpretation

In an ontology space made of autonomous independent ontologies, reasoning has to consider how to interpret definitions to which explicit correspondences have not been specified. In the running example, analyzing the satisfiability of query $Q = x_1 \subseteq x_2$, depends on the given interpretation for both x_1 and x_2 . If a local interpretation is assumed, by prefixing each ontology entity with a local identification, then satisfiability is only achieved if explicit correspondences associate query terms interpretation with ontology entities used for reasoning.

On the other hand, one may be interested in possible answers for the reasoning query. In this scenario, entities computed as relevant that present the same term are considered as having an implicitly equivalence correspondence. The motivation for such assumption is that relevant entities are taken from the intersection set of query clause with relevant correspondences (see Section 15.3.2) in the remote ontology. This reinforces that both terms share the same semantic context and, thus, may be equivalent. Producing possible answers may include providing users with a list of assumed correspondences, so that further processing may analyze its pertinence.

15.4 Applying the Reasoning Space Approach into a Use Case

In this section, we illustrate the procedure for reasoning over a *reasoning space* as presented in Section 15.3 above. We consider a use case in which users search for Web services that provide car rental services.

We take the approach presented in [KLP⁺04] in which Web service functionality (or capability in WSMO terms [LRK04]) is described by means of *conjunctive formulae* [CGL90] indicating the objects involved in the functionality provided by the Web service and relationship between these objects. Correspondingly, user queries are grounded conjunctive queries that express the desired service, which in WSMO is called a *Goal*.

Thus, finding a Web service that satisfies the user corresponds to matching the user goal against descriptions of Web service functionality. Unfortunately, very often, terminologies used in describing the goal and the web service functionality may be different. This is where the ontology space comes into the game. It provides the means to verify the correspondences between terms used in the goal and web service functionality definitions.

In this context, let us consider an ontology space $OS = \{O_1, O_2\}$, with its corresponding modules $M = \{M_1, M_2\}$ that are used by the matching algorithms to eliminate ambiguities and heterogeneities in between goal and web service description terminology.

An agent looking for booking a *sportscar* in the city of *Lausanne*, as part of a tourism package, would initiate a Web service discovery process by submitting a corresponding goal to the system. Let's assume that a single Web service has been advertised by offering as one of its functionalities the rental of a set of car models in Europe.

The agent's goal g and Web service description ws would be expressed as below:

$g = \text{carRental}$	and	$\text{model}(\text{sportscars})$	and	$\text{place}(\text{Lausanne})$
$ws = \text{carRental}$	and	$\text{model}(\text{Ferrari})$	and	$\text{place}(\text{Europe})$

Based on this input, the discovery process initiates a matching function which analyzes the correspondences between predicates *carRental*, *model* and *place* in g and ws . These, however, cannot be directly matched because of the semantic heterogeneity between the goal and the web service description. Ontological support is needed to overcome the semantic gap. Thus, the matching function submits a query to module M_1 to find out whether *Ferrari* is a model of *sportscar* and *Lausanne* is a place in *Europe*, which would lead to a successful match between the goal g and the web service description ws . The query to M_1 is expressed as:

$q: \text{Ferrari} \sqsubseteq \text{sportscar} \text{ and } \text{Lausanne} \sqsubseteq \text{Europe}$

The reasoning task is evaluated considering the module $M1 = \langle I, d, l, C_1, O_1, O_2 \rangle$, exemplified in Tables 15.1 and 15.2.

Table 15.1: Ontologies O_1 and O_2 .

O_1	O_2
Concept(Car)	Concept(vehicle)
Concept(turbo_engine_car)	Concept(sportscar)
Concept(Lausanne)	Concept(Ferrari)
Concept(EU)	Concept(Europe)
Turbo_engine_car \subseteq Car	sportscar \subseteq vehicle
Lausanne \subseteq EU	Ferrari \subseteq sportscar

Table 15.2: Ontology Correspondence Definitions c_1

c_{11} : O_1 : turbo_engine_car \supseteq O_2 : Ferrari
c_{12} : O_1 : EU \equiv O_2 : Europe

Query q is in RCF, presenting clauses $t_1 = \text{Ferrari} \subseteq \text{sportscar}$ and $t_2 = \text{Lausanne} \subseteq \text{Europe}$. The evaluation of q initially considers the *reasoning space* $RS = O_1 \cup C_1$. In this context, clause t_2 can be decided by using correspondence c_{12} , the same not being observed with respect to the clause t_1 that remains undecided. The evaluation of q proceeds by extending the initial reasoning space towards relevant entities defined in O_2 , with respect to t_1 .

The logical expression in t_1 specifies the set of objects where Ferrari is a subset of sportscar. Analyzing the set of relevant correspondences in C_1 , c_{11} is identified as providing the set of objects where Ferrari is a *turbo_engine_car*, thus $c_{11} \cap t_1 \neq \emptyset$ and is chosen to compose the set of relevant correspondences. The *relevant entities* of O_2 with respect to c_{11} is obtained by evaluating $RE = \{O_2 \cap c_{11}\} \Rightarrow \{\text{Concept(vehicle)}, \text{Concept(sportscar)}, \text{Concept(Ferrari)}, \text{Ferrari} \subseteq \text{sportscar}, \text{sportscar} \subseteq \text{vehicle}\}$.

Finally, the reasoning space RS is augmented with $RS = RS \cup RE$ and the evaluation of t_1 can take place.

An attentive reader may argue that the query rewriting approach [8] could be used to decide on query q without the burden of formulating a global RS . This would be the case if we could guarantee consistency over ontologies in the ontology space. As discussed in Section 15.1, conflicting definitions among participating ontologies may raise as a result of autonomous ontology evolution. In this context, if queries are rewritten and evaluated over single ontologies, such conflicts would be impossible to detect, bringing eventually

to users contradictory answers, which justifies the proposed approach for reasoning over a single logical theory that is incrementally extended.

15.5 Conclusion

Reasoning over distributed and heterogeneous ontologies is not an easy task. First, there are no currently available distributed reasoners. Second, keeping correspondences between ontology entities up to date is hard as ontologies evolve. Third, as ontologies cover more complex domains their size augments precluding a complete transfer of whole ontologies to the queried peer. Finally, inconsistencies among ontologies may offer users contradictory answer that would be hard to detect once the whole result has been produced.

In this contribution, we presented a strategy for reasoning over a set of autonomously managed ontologies with correspondences defining local interpretations for foreign defined ontology entities. In our approach, a reasoning space is built including relevant ontology entities, with respect to a ontology query, found in foreign ontologies. Relevant entities are obtained by computing intersections among ontology entities and query clauses. Entities thus after discovered fill the reasoning space allowing the use of efficient and available reasoner tools.

The approach presents solutions to all identified problems but also brings to light new questions. As a matter of fact, deciding on inconsistencies on such an autonomous settings is not easy as it has been discussed with respect to non-explicit correspondences. Clearly, a more precise comparison of our approach with other distributed ontology reasoning based on query rewriting is of primordial importance to evaluate the benefits of building a reasoning space. This is in our list of future work. We also plan to implement our approach in a P2P system developed in the context of the DIP project. Finally, we also want to investigate a cost model for expanding the reasoning space. The main intuition is that there are innumerable equivalent paths to follow in exploring the ontology space. A cost model based on previous reasoning tasks and statistics regarding individual ontology entities should certainly contribute to reduce the query elapsed-time.

Chapter 16

Decentralized Case-Based Reasoning with an Application to Oncology

by MATHIEU D'AQUIN, JEAN LIEBER, AMEDEO NAPOLI

Ontology modularization is generally concerned with the decomposition of the knowledge about a domain in several parts, called modules, that are considered to be significant with respect to this domain. Centralized reasoning systems are then replaced by distributed mechanisms for reasoning over modular ontologies. In order to be efficient, these mechanisms exploit the distribution of the knowledge into modules and, as well, the relations between these modules. This chapter addresses the issue of the practical use of modular ontologies for a given domain: oncology. Case-based reasoning (CBR, see e.g. [LBSBW98, AP94]) is the reasoning methodology used for decision support in this framework. Following the principle of decentralized artificial intelligence [DM89], we propose a decentralized CBR (DzCBR) mechanism based on modular ontologies in the C-OWL formalism [BGvH⁺04].

16.1 Introduction and Motivation: Adaptation Within Multiple Viewpoints in Oncology

Oncology is a complex domain where several specialties, e.g. chemotherapy, surgery and radiotherapy, are involved in several treatment phases. In most cases, the adequate therapeutic decision is given according to a protocol that associates standard patient characteristics with a recommended treatment. Even if it is designed to take into account the majority of the medical cases, a protocol does not cover all the situations. Decisions concerning patients out of the protocol are elaborated within a multi-disciplinary expert committee, and rely on the adaptation of the solutions provided by the protocol for similar cases. Furthermore, specialties in oncology organize their background knowledge and past experiences in different ways. Indeed, a protocol is structured according to the on-

ology specialties and, during a meeting of an expert committee, each expert from each specialty supplies a personal view on the solution as a part of a collective solution. For each specialty, a particular type of treatment is requested, in a particular treatment phase, and the patient characteristics used to elaborate the solution change from a specialty to another. Thus, oncology specialties provide different viewpoints on oncology, and these viewpoints are related to each other. Information about a problem, e.g. finding a therapeutic decision for a patient, can be shared across specialties, and decisions taken in a particular specialty may influence decisions taken in another one.

CBR is a type of analogical reasoning in which problem-solving is based on the adaptation of the solutions of similar problems, already solved and stored in a case base. In particular, knowledge-intensive CBR (KI-CBR [Aam04]) relies on a knowledge base including domain knowledge and, as well, knowledge units exploited for the retrieval and adaptation operations of CBR (called *adaptation knowledge* hereafter). In the perspective of decision support for out of the protocol cases, a KI-CBR mechanism relying on a formalized protocol may be applied. In this way, the knowledge used by expert committees is represented and operationalized in the form of adaptation knowledge to become sharable and reusable.

C-OWL (for context-OWL) is a formalism that has been proposed by [BGvH⁺04] for the representation of mappings between several OWL ontologies. A local ontology in C-OWL is considered as a context, having its own language and its own interpretation. Mappings are made of bridge rules that express semantic relations between classes, properties and individuals of the local ontologies. In this way, mappings between ontologies using C-OWL allow the coordinate use of these ontologies, keeping the knowledge contained in each of them in its local context. Then, C-OWL allows us to represent modular ontologies for combining the multiple viewpoints involved in oncology, and a KI-CBR mechanism may be used with profit for exploiting such decentralized knowledge. The framework of DzCBR is proposed here for this purpose. In DzCBR, several CBR processes are carried out, each of them exploiting domain knowledge and adaptation knowledge locally in a particular context, treating the problem according to a particular viewpoint. Collaboration between viewpoints is then achieved thanks to bridge rules between contexts, on the basis of global reasoning in distributed description logic as studied in [ST05] and reported in Chapter 14.

The next section contains a brief introduction to CBR. It also indicates how CBR is integrated within the semantic Web framework using OWL. The Section 16.3 details the knowledge and reasoning models of DzCBR, and how problem-solving is carried out by combining several decentralized viewpoints represented by C-OWL contexts. An application of DzCBR to a breast cancer treatment problem is presented in Section 16.4. Finally, the related work is discussed in Section 16.5.

16.2 Case-Based Reasoning with OWL

16.2.1 Principles of Case-Based Reasoning

A case is a problem solving episode usually represented by a *problem* pb and a *solution* $Sol(pb)$ of pb . A case base is a (usually structured) set of cases, called *source cases*. A source case is denoted by $(srce, Sol(srce))$. CBR consists in solving a *target problem*, denoted by tgt , thanks to the case base. The classical CBR process relies on two steps, retrieval and adaptation. *Retrieval* aims at finding a source problem $srce$ in the case base that is considered to be similar to tgt . The role of the *adaptation* task is to adapt the solution of $srce$, $Sol(srce)$, in order to build $Sol(tgt)$, a solution of tgt . Then, the solution $Sol(tgt)$ is tested, repaired, and, if necessary, memorized for future reuse.

In knowledge intensive CBR (KI-CBR, see e.g. [Aam04, GAGCDAFC99, LN98]), the CBR process relies on a formalized model of domain knowledge. This model may contain, for example, an ontology of the application domain, and can be used to organize the case base for case retrieval. KI-CBR may also include some knowledge for adaptation, as explained in the following.

16.2.2 Reformulations: an Approach for Representing Adaptation Knowledge

Reformulations are basic elements for modeling adaptation knowledge for CBR [MLN98]. A reformulation is a pair (r, \mathcal{A}_r) where r is a relation between problems and \mathcal{A}_r is an *adaptation function*: if r relates $srce$ to tgt —denoted by “ $srce \ r \ tgt$ ”—then any solution $Sol(srce)$ of $srce$ can be adapted into a solution $Sol(tgt)$ of tgt thanks to the adaptation function \mathcal{A}_r —denoted by “ $Sol(srce) \ \mathcal{A}_r \ Sol(tgt)$ ”.

In the reformulation model, retrieval consists in finding a *similarity path* relating $srce$ to tgt , i.e. a composition of relations r_k , introducing intermediate problems pb_k between the source and the target problems. Every r_k relation is linked by a reformulation to an adaptation function \mathcal{A}_{r_k} . Thus, the sequence of adaptation functions following the similarity path may be reified in an *adaptation path* (see figure 16.1).

The model of reformulations is a general framework for representing adaptation knowledge. The operations corresponding to problem relations r_k and adaptation functions \mathcal{A}_{r_k} have to be designed for a particular application. Generally, these operations rely on transformation operations such as specialization, generalization and substitution, that allow the creation of the pb_k problems for building the similarity path and of the $Sol(pb_k)$ solutions for the adaptation path: relations of the form $pb_1 \ r \ pb_2$ and adaptation like $Sol(pb_1) \ \mathcal{A}_r \ Sol(pb_2)$ correspond to applications of such transformations.

Moreover, the reformulation framework follows the principle of adaptation-guided re-

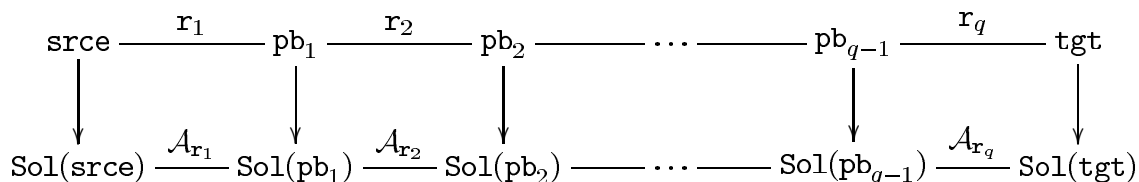


Figure 16.1: A similarity path from $srce$ to tgt (first line) and the corresponding adaptation path (second line).

trieval [Smy96]. A CBR system using adaptation-guided retrieval retrieves the source cases whose solution is adaptable, i.e. for which adaptation knowledge is available. According to this principle, similarity paths provide a kind of symbolic reification of similarity between problems, allowing the case-based reasoner to build understandable explanation of the results.

16.2.3 CBR within OWL ontologies

In OWL, problems and solutions are represented as instances of the `Problem` and the `Solution` classes. The link between a problem pb and its solution $Sol(pb)$ is materialized by a property called `hasSolution`. OWL axioms are used to relate `Problem` and `Solution` to classes of the domain knowledge. For example, in an application for breast cancer treatment, the `Patient` and `Treatment` classes correspond respectively to the `Problem` and `Solution` classes, and thus, the two axioms $Patient \sqsubseteq Problem$ and $Treatment \sqsubseteq Solution$ are added to the ontology. Furthermore, the `hasSolution` property relates patients to the recommended treatments. Problem relations, adaptation functions and reformulations are also formalized in OWL. The specific underlying mechanisms are made by Web services implementing transformation operations like specialization, generalization and property substitution on OWL individuals.

Given two classes C and D , the *subsumption test* in OWL is defined by C is subsumed by D (C is more specific than D) if, for every model \mathcal{I} of O , $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. Based on the subsumption test, *classification* consists in finding for a class C , the most specific classes in the ontology subsuming C , and the most general classes subsumed by C . Classification organizes the classes of the ontology in a hierarchy. Regarding CBR, the class hierarchy is used as a structure for the case base, where a class represents an index for a source problem. Every index is considered as an abstraction of a source problem, containing the relevant part of the information leading to a particular solution.

Instance checking tests whether an individual a is an instance of a class C , i.e. if for every model \mathcal{I} of O , $a^{\mathcal{I}} \in C^{\mathcal{I}}$. It supports the *instantiation* reasoning service that consists in finding the most specific classes of an individual. It is used during the retrieval step of CBR for finding index classes of source problems. A source problem $srce$ is an instance of its index class $idx(srce)$, and its solution $Sol(srce)$ is considered to be reusable

for any problem pb that is an instance of $idx(srce)$, i.e. $Sol(srce)$ can be reused to solve tgt whenever tgt is recognized as an instance of $idx(srce)$.

Instantiation is used to infer new pieces of information about an individual on the basis of its class membership, and of constraints contained in class definitions. For example, if an individual named bob is an instance of the class Man , if Man is declared to be more specific than $Human$ ($Man \sqsubseteq Human$), and if the capability of intelligence is associated with humans ($Human \sqsubseteq \exists capability.Intelligence$), then, bob has to be capable of intelligence. The information known about bob is automatically completed, thanks to constraints inherited from $Human$. This reasoning service has proved to be useful for CBR in [GAGCDAFC99], where it is called *instance completion*. Particularly, it is used in the *problem elaboration* operation, to extend the available information on the target problem with respect to the domain knowledge. Moreover, since a particular index $idx(srce)$ may lead to a particular solution $Sol(srce)$, this solution can be directly attached to the index class through a *problem-solution axiom* of the form: $I \sqsubseteq \exists hasSolution.S$. This means that, based on instance completion, any instance of the index class I is related to an object of the solution class S by the `hasSolution` property.

16.3 Decentralized Case-Based Reasoning with C-OWL

Decentralized artificial intelligence, as defined by [DM89], is concerned with the activity of autonomous intelligent agents that coexist and may collaborate with other agents, each of them having its own goals and its own knowledge. In the same way, the DzCBR mechanism is:

1. local to a context in the sense that it is carried out in each context, not in a centralized manner,
2. collaborative in the sense that it relies on knowledge sharing between contexts.

16.3.1 CBR with Contextualized Knowledge

Contextualized ontologies in C-OWL are local representations of a domain, named contexts, that are semantically related with other contexts thanks to mappings [BGvH⁺04]. In C-OWL, the knowledge about a domain is contained in a set of contexts, called a *context space*. Each context O_i of a context space is an OWL ontology, with its own language and its own interpretation. Mappings are expressed by bridge rules that are used to declare correspondences between the interpretation domains of two different contexts. An *into rule* is a bridge rule of the form $i:C \xrightarrow{\sqsubseteq} j:D$, where $i:C$ and $j:D$ are classes respectively from O_i and O_j . This type of rule means that the class $i:C$ of O_i is considered,

from the viewpoint of O_j , to be more specific than the class $j:D$ [ST05]. The *onto rule* $i:C \xrightarrow{\sqsupseteq} j:D$ means that O_j considers the class $i:C$ to be more general than $j:D$. In addition to the basic into and onto rules between classes, we also use another form of bridge rules for specifying correspondences between individuals. $i:a \xrightarrow{\equiv} j:b$ means that, according to O_j , the individual $i:a$ in O_i corresponds to the individual $j:b$.

Using C-OWL for DzCBR, a context is used to represent a particular viewpoint on the domain. A global target problem is represented by a set $\{i:tgt\}_i$ of local target problems, with a problem $i:tgt$ in each context O_i . In addition, a bridge rule $i:tgt \xrightarrow{\equiv} j:tgt$ is declared for each O_i and O_j of the context space, i.e. $i:tgt$ in O_i is viewed as $j:tgt$ in O_j .

A context O_i includes knowledge and cases which allows to find a local solution $i:Sol(tgt)$ for the local problem $i:tgt$. Thus, a local problem $i:pb$ is solved by a solution $i:Sol(pb)$ inside the context O_i . The adaptation knowledge used for solving a local problem $i:tgt$ is also represented within the context O_i . Local reformulations $i:(r, \mathcal{A}_r)$ are the basic adaptation knowledge units for solving $i:tgt$ in the O_i context.

In a context O_i , there is a class hierarchy where a class represents the index of a source problem to be reused. An index $i:idx(srce)$ is an abstraction of the problem $i:srce$, retaining the relevant information according to the viewpoint of the O_i context, i.e. $i:Sol(srce)$ can be reused to solve $i:tgt$ whenever $i:tgt$ is an instance of $i:idx(srce)$ (in accordance with the solving schema described in the section 16.2.3).

Then, in O_i , the instantiation reasoning service is used in a *localized retrieval* process for finding the index $i:idx(srce)$ of the source problem $i:srce$ to be reused. More precisely, the retrieval process consists in finding a similarity path between the target problem $i:tgt$ and the index $i:idx(srce)$ that is composed of relations defined in O_i :

$$i:srce \xrightarrow{isa} i:idx(srce) \xleftarrow{isa} i:pb_1 \ i:r_1 \ \dots \ i:r_q \ i:tgt$$

where the “isa” arrows mean “is an instance of”. In addition, a *localized adaptation* process has to build an associated adaptation path using reformulations and adaptation functions defined in O_i for building $i:Sol(tgt)$. Using contextualized knowledge and cases, the CBR process is then “contained” in a context. A detailed example of this localized CBR process is given at the end of the next section.

16.3.2 Combining Viewpoints Thanks to Bridge Rules

[ST05] presents an extension of the standard tableau algorithm for the computation of the global subsumption test. *Global subsumption* uses the principle of subsumption propaga-

tion which, in its simplest form, can be expressed as:

if the mapping \mathcal{M}_{ij} contains $i:A \xrightarrow{\exists} j:C$ and $i:B \xrightarrow{\sqsubseteq} j:D$
 then \mathcal{I} satisfies $i:A \sqsubseteq B$ implies that \mathcal{I} satisfies $j:C \sqsubseteq D$.

where \mathcal{I} is a distributed interpretation containing a local interpretation for each context of the context space and semantic relations for interpreting bridge rules. Intuitively, this means that subsumption in a particular context can be inferred from subsumption in another context thanks to bridge rules. Similarly, we consider here a *global instance checking* based on an instantiation propagation rule:

if \mathcal{M}_{ij} contains $i:C \xrightarrow{\sqsubseteq} j:D$ and $i:a \xrightarrow{\equiv} j:b$
 then \mathcal{I} satisfies $i:C(a)$ implies that \mathcal{I} satisfies $j:D(b)$.

Instantiation is extended in order to use global instance checking. Based on bridge rules, information known about an individual in a particular context can be completed using inferences made in another context.

In the following, we present an example of a DzCBR process that is distributed among contexts and that takes advantage of this distribution for building a global solution for a target problem.

Let us introduce three contexts named O_1 , O_2 and O_3 , where a source problem is represented by its index class, and each association between a problem and its solution is represented by a problem-solution axiom. For example, the expression $1:I1 \equiv \text{Problem} \sqcap \exists p1.C1$ defines a source problem in the context O_1 , and $1:I1 \sqsubseteq \exists \text{hasSolution}.S1$ associates an instance of the solution class $1:S1$ to an instance of the problem class $1:I1$. In the same way, the source problems $2:I2$ and $3:I3$ are respectively defined in the contexts O_2 and O_3 , together with their problem-solution axioms (1st and 2nd lines of the figure 16.2). Bridge rules have been declared between the three local target problems $1:tgt$, $2:tgt$ and $3:tgt$, making precise the fact that these local problems are three views about a single problem (3rd line of the figure 16.2). Moreover, bridge rules between classes indicate the subsumption constraints between the contexts (4th line of the figure 16.2). Finally, a set of assertions is given for the three local target problems (5th, 6th and 7th lines of figure 16.2).

When the DzCBR process is run in each context, the three local target problems $1:tgt$, $2:tgt$, and $3:tgt$ are instantiated in their respective contexts.

Dz1. In the O_2 context, $2:tgt$ is recognized as an instance of the class $2:\exists p21.C21$.

Dz2. The bridge rules $2:\exists p21.C21 \xrightarrow{\sqsubseteq} 1:\exists p1.C1$ and $2:tgt \xrightarrow{\equiv} 1:tgt$ allow the completion of the instance $1:tgt$. $1:tgt$ is recognized as an instance of the class $1:\exists p1.C1$, and thus of the class $1:I1$.

O_1	O_2	O_3
$I1 \equiv \text{Problem} \sqcap \exists p1.C1$ $I1 \sqsubseteq \exists \text{hasSolution}.S1$	$I2 \equiv \text{Problem} \sqcap \exists p21.C21 \sqcap \exists p22.C22$ $I2 \sqsubseteq \exists \text{hasSolution}.S21$	$I3 \equiv \text{Problem} \sqcap \exists p3.C3$ $I3 \sqsubseteq \exists \text{hasSolution}.S31$
$2:\text{tgt} \xrightarrow{\equiv} 1:\text{tgt}$ $2:\exists p21.C21 \xrightarrow{\sqsubseteq} 1:\exists p1.C1$	$1:\text{tgt} \xrightarrow{\equiv} 2:\text{tgt}$ $1:\exists \text{hasSolution}.S1 \xrightarrow{\sqsubseteq} 2:\exists p23.C23$	$2:\text{tgt} \xrightarrow{\equiv} 3:\text{tgt}$ $2:\exists \text{hasSolution}.S22 \xrightarrow{\sqsubseteq} 3:\exists \text{hasSolution}.S32$
Problem(tgt)	Problem(tgt) C21(a) p21(tgt, a)	Problem(tgt)
Dz2. $\exists p1.C1(\text{tgt})$ Dz3. $\exists \text{hasSolution}.S1(\text{tgt})$	Dz1. $\exists p21.C21(\text{tgt})$ Dz4. $\exists p23.C23(\text{tgt})$ Dz5. $\exists \text{hasSolution}.S22(\text{tgt})$	Dz6. $\exists \text{hasSolution}.S32(\text{tgt})$

Figure 16.2: A DzCBR example. 1st and 2nd lines define some sources problems. 3rd and 4th lines describe mappings associated with the contexts. 5th to 7th lines describe the target problem. 8th to 11th lines show 6 DzCBR inference steps.

- Dz3. Through the problem-solution axiom, a solution $1:S1$ is associated with $1:\text{tgt}$, that in turn becomes an instance of the class $1:\exists \text{hasSolution}.S1$.
- Dz4. The instance completion process is run through the bridge rule $1:\exists \text{hasSolution}.S1 \xrightarrow{\sqsubseteq} 2:\exists p23.C23$, and the local target problem $2:\text{tgt}$ is recognized as an instance of the class $2:\exists p23.C23$.
- Dz5. As it is explained below, let us assume that the CBR process in the context O_2 builds a solution that is an instance of $2:S22$ and that is associated with $2:\text{tgt}$. $2:\text{tgt}$ becomes an instance of $2:\exists \text{hasSolution}.S22$ in O_2 .
- Dz6. Finally, based on the bridge rule $2:\exists \text{hasSolution}.S22 \xrightarrow{\sqsubseteq} 3:\exists \text{hasSolution}.S32$, it can be inferred in O_3 that $3:\text{tgt}$ is an instance of $3:\exists \text{hasSolution}.S32$.

The solution of the target problem, represented by the three local target problems $1:\text{tgt}$, $2:\text{tgt}$, and $3:\text{tgt}$, is a set of local solutions, represented as instances of $1:S1$, $2:S22$, and $3:S32$, that have been built in a decentralized way.

Relying on this example, two main operations may be distinguished in the DzCBR process:

- (i) *localized CBR* that applies local knowledge for building a solution to the local problem $i:\text{tgt}$. The steps Dz3. and Dz5. are examples of such a local operation in DzCBR, respectively carried out in O_1 and O_2 .
- (ii) *case completion* represents the *collaborative* part of DzCBR. It is based on bridge rules and completes the local target case –either the problem or the solution part–

thanks to knowledge sharing with the other contexts. The steps Dz2., Dz4. and Dz6. are examples of this collaboration, using bridge rules for combining viewpoints.

These two operations are run in each context, until no more inferences can be drawn. The solution set $\{i : \text{Sol}(tgt)\}_i$ is then delivered.

16.3.2.1 Details of the localized CBR Process Dz5.

The O_2 context contains a reformulation of the form $2 : (r, \mathcal{A}_r)$ that is used in the localized CBR operation in this context (see figure 16.3). During the retrieval step, the $2 : r$ relation creates an intermediary problem $2 : pb_1$ from $2 : tgt$ such that the difference between these two individuals lies in the fact that $2 : pb_1$ is an instance of $2 : \exists p22.C22$, whereas $2 : tgt$ is an instance of $2 : \exists p23.C23$. Thus, $2 : pb_1$ is recognized as an instance of $2 : I2$, and is associated with a solution $2 : \text{Sol}(pb_1)$ from $2 : S21$, as stated by the problem-solution axiom in O_2 . The $2 : \mathcal{A}_r$ adaptation function is used in the adaptation step for creating the solution $2 : \text{Sol}(tgt)$ from $2 : \text{Sol}(pb_1)$. $2 : \mathcal{A}_r$ is such that the difference between $2 : \text{Sol}(pb_1)$ and $2 : \text{Sol}(tgt)$ lies in the fact that $2 : \text{Sol}(pb_1)$ is an instance of $2 : S21$, whereas $2 : \text{Sol}(tgt)$ is an instance of $2 : S22$. Therefore, $2 : \text{Sol}(tgt)$, instance of $2 : S22$, becomes a solution of $2 : tgt$.

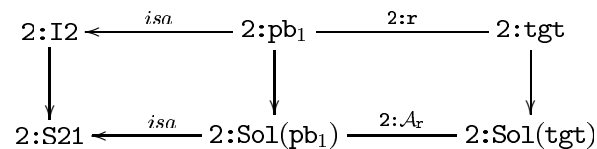


Figure 16.3: The similarity path and the adaptation path of the localized CBR process in O_2 .

16.4 Application to Breast Cancer Treatment

The task of finding the right treatment for a patient ill with breast cancer is supported by a protocol. This protocol can be seen as a set of rules $Cond \Rightarrow Ttt$ where $Cond$ is a set of conditions on patients and Ttt is a description of the type of treatments recommended for the patients satisfying $Cond$. Several specialties are involved in this decision, and the protocol is structured according to these specialties. The protocol rules may be directly applied in 60 to 70 % of the situations (with respect to the characteristics of the patients). In situations not considered by the protocol, the decision is taken by a multi-disciplinary expert committee. This committee adapts the protocol rules to find a solution, taking into account the characteristics of the considered patient.

In our research work, decision support for breast cancer treatment relies on DzCBR, where a problem is a description of the characteristics of a patient, and a solution is a treatment proposition. The case base and the domain model rely on a formalized representation of the protocol in C-OWL. In the following example, three different contexts, namely O_r , O_s , and O_c , standing for the radiotherapy, surgery and chemotherapy viewpoints, are considered. These contexts correspond respectively to the O_1 , O_2 and O_3 contexts of the example of section 16.3.2. A protocol rule $Cond \Rightarrow Ttt$ is represented and implemented as a problem-solution axiom of the form $PC \sqsubseteq \exists hasSolution.T$, where PC and T are classes respectively representing the $Cond$ and Ttt parts of the protocol rule. For example, O_r contains a problem class corresponding to the patients having a tumor that is smaller than 4cm. For the members of this class, a radiotherapy of the internal mammary chain is recommended. Therefore, the problem solution axiom $1:I1 \sqsubseteq \exists hasSolution.S1$ of the preceding example is restated as :

$$r:Patient \sqcap \exists tumorSize.lessThan4cm \sqsubseteq \exists hasSolution.IntMamChainRadio$$

In the same way, O_s contains the problem-solution axiom :

$$s:Patient \sqcap \exists hasTumor.(\exists size.moreThan4cm) \sqcap \exists radiotherapy.IntMamChain \sqsubseteq \exists hasSolution.TotalAblation$$

meaning that, for patients having a tumor greater than 4cm and for whom a radiotherapy of the internal mammary chain may be applied, a total ablation of the breast is recommended. In O_c , the axiom:

$$c:Patient \sqcap \exists lymphNode.infected \sqsubseteq \exists hasSolution.PreSurgicalChemo$$

means that for patients having infected lymph nodes, some cures of chemotherapy should be applied before the surgical treatment in order to prepare the patient for a partial ablation.

The bridge rules of the example of the section 16.3.2 are now redefined on the classes of O_r , O_s and O_c :

$$s:\exists hasTumor.(\exists size.lessThan4cm) \xrightarrow{\sqsubseteq} r:\exists tumorSize.lessThan4cm$$

$$r:\exists hasSolution.IntMamChainRadio \xrightarrow{\sqsubseteq} s:\exists radiotherapy.IntMamChain$$

$$s:\exists hasSolution.TotalAblation \xrightarrow{\sqsubseteq} c:\neg \exists hasSolution.PreSurgicalChemo$$

The first one allows the surgery context to share the information about the size of the tumor with the radiotherapy context. Problem-solving in surgery can reuse the solution found in radiotherapy thanks to the second bridge rule. The third bridge rule expresses that, when a total ablation is recommended, a chemotherapy must not be applied before surgery.

Moreover, the O_s context contains some adaptation knowledge in the form of a reformulation $s:(r, \mathcal{A}_r)$. The $s:r$ relation holds between an instance of $Patient$ having a little-sized tumor (less than 4 cm) that covers a large part of the breast (more than 60%)

and an instance of `Patient` having a larger tumor (more than 4cm). In other terms, a patient with a small tumor in a small breast is considered *for surgery* to be similar to a patient having a large tumor. The $s:\mathcal{A}_r$ adaptation function simply consists in a copy of the solution.

The target problem is represented by three local target problems denoted by $r:tgt$, $s:tgt$ and $c:tgt$, that are linked by bridge rules. Each of these individuals is an instance of the patient class, i.e. the assertions $r:Patient(tgt)$, $s:Patient(tgt)$ and $c:Patient(tgt)$ are stated in the O_r , O_s and O_c contexts respectively. Moreover, $s:tgt$ is described as a patient having a small tumor in a small breast, i.e. the assertion $s:\exists hasTumor.(\exists size.lessThan4cm \sqcap \exists cover.MoreThan60\%)(tgt)$ is stated in O_s .

The DzCBR process for solving this problem corresponds to the six steps of the section 16.3.2 example. The information about the tumor size is first shared between surgery and radiotherapy, and so, a radiotherapy of the internal mammary chain is recommended in O_r . In O_s , the reformulation $s:(r, \mathcal{A}_r)$ is applied, considering $s:tgt$ as similar to a patient having a large tumor. According to the problem-solution axiom contained in O_s , the solution for a patient with a large tumor is a total ablation. This solution is copied through \mathcal{A}_r for $s:tgt$. Finally the solution found in surgery, the total ablation, implies that no chemotherapy has to be applied before surgery. It must be remarked that the target problem is treated differently in O_s and O_r . Indeed, it has been considered as a patient with a small tumor for radiotherapy, whereas it has been considered as a patient with a large tumor in surgery.

16.5 Discussion and Related Work

A CBR system based on the reformulation model has been implemented in the form of a generic Web service manipulating OWL ontologies. This architecture based on Web services is very helpful in the implementation of localized CBR. For global reasoning in C-OWL, we are using the system described in [ST05] that is currently under development. A complete protocol for breast cancer treatment has also been formalized in C-OWL. The lesson learned from this experiment is that building and managing multiple contexts that reflect existing viewpoints in the domain appear to be simpler than finding and maintaining a consensual representation for these viewpoints all together.

Considering related work, description logics have been used for KI-CBR in several systems (see e.g. [GAGCDAFC99, KLG98]). These systems consider a single knowledge model, and take into account a single way of interpreting and using cases. DzCBR combines several viewpoints on the problems and solutions, thanks to multiple inter-related contexts. Some systems use several views on cases to retrieve several local best cases. Generally, a single global case is build from these sub-cases. For example, in [AL97] a choice is made between cases that are retrieved using different case representations, called

perspectives. In [NLL96], several agents retrieve local best cases that are assembled in a global best case thanks to negotiation between agents. Since there is no centralized mechanism in DzCBR, a CBR process is carried out in each context and collaborates with the other contexts through bridge rules. In this way, among contexts, several local source cases are retrieved and used independently for adaptation.

Our interest for a DzCBR process exploiting semantic Web technologies and principles has started with the design of a semantic portal for oncology [dBB⁺05]. The goal of this portal is to give an intelligent access to standard knowledge for a geographically distributed community of oncologists. There are many other situations, like adaptive query answering, case-based ontology alignment or flexible Web service invocation, where CBR would be useful for the semantic Web. Some studies have been interested in defining markup languages for case representation, on the basis of XML [CDC04] or RDF [CW03]. But, to our knowledge, there are no works concerned with the design of CBR systems in the semantic Web framework.

Chapter 17

Conclusion

This deliverable reported on the continued effort by WP 2.1 to clarify the issues related to ontology modularization and explore the research directions that aim at making the modularization idea operational. Clarification was felt necessary as ontology modularization is a new problem, only recently tackled, and many different paths are being explored, with no single dominant approach emerging. This deliverable has explored many of the potential avenues, and should be seen as explorative research.

The first part of the deliverable aims at making clear that there are alternative perceptions of what ontology modularization means. In particular, we identified a composition versus a decomposition approach to modularization. In the former, a set of existing source ontologies are apprehended as modules of a larger ontology that is built from the pre-existing sources using some integration technique. In the latter, it is the global ontology that pre-exists, and modularization is seen as the process of producing a consistent set of sub-ontologies, the modules, using some decomposition technique.

Beyond this major split within the set of approaches that deal with modularization, we identified a number of issues, from the precise definition of the module concept to how different modules can be interconnected to show complementarities of their semantic content.

The second part of the deliverable is devoted to various presentations of different techniques that in one way or another contribute to modularization. These techniques range from those addressing design of modular ontologies to those supporting reasoning over a set of distributed ontologies.

The major result of the reported work, beyond the deliverable itself, is the increased awareness from the participants about the multiplicity of viewpoints on modularization, resulting in an effort by each partner to identify the exact assumptions that underlie each technique so that misunderstanding is avoided. Moreover, several partners have been able to point at connections between different works, either in terms of similarities or in terms of complementarities. This paves the way to fruitful cooperation in the future.

As future work is concerned, each partner has precise plans for the continuation of the work in the direction they have undertaken. An additional research direction will complement the ongoing researches described in previous chapters. Its focus is on using fuzzy techniques to support the possibility to attach different confidence degrees to the mappings between ontological modules, thus leading to some form of fuzzy ontology. In summary, we believe the NoE is fully playing its role. Moreover, although it is premature to talk about future joint work, we are confident that commonalities of interest will arise and lead indeed to some joint work.

Bibliography

- [Aam04] A. Aamodt. Knowledge-Intensive Case-Based Reasoning in CREEK. In P. Funk and P. A. González-Calero, editors, *Proc. of the European Conference on Case-Based Reasoning, ECCBR'04*, volume 3155 of *Lecture Notes in Artificial Intelligence*, pages 1–15. Springer, 2004.
- [AFL99] Nicolas Anquetil, Cédric Fourrier, and Timothy C. Lethbridge. Experiments with hierarchical clustering algorithms as software modularization methods. In *Proceedings of the Working Conference on Reverse Engineering (WCRE'99)*, pages 304–313, Atlanta, USA, oct 1999.
- [AL97] J. L. Arcos and R. Lopez de Mántaras. Perspectives: a declarative bias mechanism for case retrieval. In D. B. Leake and E. Plaza, editors, *Proc. of the International Conference on Case-Based Reasoning, ICCBR'97*, volume 1266 of *Lecture Notes in Computer Science*, pages 279–290. Springer, 1997.
- [AM05] E. Amir and S. McIlraith. Partition-based logical reasoning for first-order and propositional theories. *Artificial Intelligence*, 2005. Accepted for Publication.
- [AP94] Agnar Aamodt and Enric Plaza. Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. *AICom - Artificial Intelligence Communications*, 7(1):39–59, 1994.
- [Bat03] V. Batagelj. Analysis of large networks - islands. Presented at Dagstuhl seminar 03361: Algorithmic Aspects of Large and Complex Networks, August/September 2003.
- [BBDD97] L.C. Briand, C. Bunse, J.W. Daly, and C. Differding. An experimental comparison of the maintainability of object-oriented and structured design documents. *Empirical Software Engineering*, 2(3):291–312, 1997.
- [BCG04] E. Sirin B. Cuenca Grau, B. Parsia. Working with multiple ontologies on the semantic web. In S. A. McIlraith, D. Plexousakis, and

- F. van Harmelen, editors, *Proceedings of the Third International Semantic Web Conference (ISWC 2004)*, volume 3298 of *Lecture Notes in Computer Science*, pages 620–634. Springer Verlag, 2004.
- [BCM⁺03] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P.F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.
- [BEE⁺05] Paolo Bouquet, Marc Ehrig, Jérôme Euzenat, Enrico Franconi, Pascal Hitzler, Markus Krötzsch, Luciano Serafini, Giorgos Stamou, York Sure, and Sergio Tessaris. Specification of a common framework for characterizing alignment. Technical report, KnowledgeWeb Deliverable, 2005.
- [BGH99] L. Bird, A. Goodchild, and T. Halpin. Object role modelling and xml-schema. In A. Laender, S. Liddle, and V. Storey, editors, *Proceedings of the 19th International Conference on Conceptual Modeling (ER00)*, LNCS. Springer Verlag, 1999.
- [BGv⁺04] P. Bouquet, F. Giunchiglia, F. van Harmelen, L. Serafini, and H. Stuckenschmidt. Contextualizing ontologies. *Journal on Web Semantics*, 1(4):325–343, 2004.
- [BGvH⁺03] P. Bouquet, F. Giunchiglia, F. van Harmelen, L. Serafini, and H. Stuckenschmidt. C-OWL: Contextualizing Ontologies. In *Proceedings of the 2d International Semantic Web Conference (ISWC 2003)*, pages 164–179, 2003.
- [BGvH⁺04] P. Bouquet, F. Giunchiglia, F. van Harmelen, L. Serafini, and H. Stuckenschmidt. Contextualizing Ontologies. *Journal of Web Semantics*, 1(4):1–19, 2004.
- [BKvH02] Jeen Broekstra, Arjohn Kampman, and Frank van Harmelen. Sesame: A generic architecture for storing and querying rdf and rdf schema. In Ian Horrocks and James Hendler, editors, *Proceedings of the First International Semantic Web Conference (ISWC 2002)*, pages 54–68, Sardinia, Italy, jun 2002.
- [BLRT05] B. Benatallah, M. Hacid Al. Leger, C. Rey, and F. Toumani. On automating web service description. *VLDB Journal*, 14:84–96, 2005.
- [BM03] V. Batagelj and A. Mrvar. Pajek - analysis and visualization of large networks. In M. Juenger and P. Mutzel, editors, *Graph Drawing Software*, pages 77–103. Springer, 2003.

- [BS03] A. Borgida and L. Serafini. Distributed description logics: Assimilating information from peer sources. *Journal of Data Semantics*, 1:153–184, 2003.
- [Bur92] R.S. Burt. *Structural Holes. The Social Structure of Competition*. Harvard University Press, 1992.
- [BvHH⁺04] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D.L. McGuinness, P.F. Patel-Schneider, and L. Andrea Stein. OWL Web Ontology Language Reference. W3C Recommendation, February 2004. www.w3.org/TR/owl-ref.
- [CDC04] L. Coyle, D. Doyle, and P. Cunningham. Representing Similarity for CBR in XML. In P. Funk and P. González Calero, editors, *Advances in Case-Based Reasoning (Procs. of the Seventh European Conference)*, *LNAI 3155*, pages 119–127. Springer, 2004.
- [CGL90] S. Ceri, G. Gottlob, and L. Tanca. *Logic Programming and Databases*. Springer-Verlag, 1990.
- [CW03] H. Chen and Z. Wu. CaseML: a RDF-based Case Markup Language for Case-based Reasoning in Semantic Web. In B. Fuchs and A. MilLe, editors, *From structured cases to unstructured problem solving episodes for experience-based assistance. Workshop at ICCBR-2003*, 2003.
- [dBB⁺05] M. d’Aquin, S. Brachais, C. Bouthier, J. Lieber, and A. Napoli. Knowledge Editing and Maintenance Tools for a Semantic Portal in Oncology. *International Journal of Human-Computer Studies (IJHCS)*, 2005. To appear. Also available online <http://authors.elsevier.com/sd/article/S1071581905000285>.
- [DJM02] Jan Demey, Mustafa Jarrar, and Robert Meersman. A Conceptual Markup Language That Supports Interoperability between Business Rule Modeling Systems. In *Proceedings of the Tenth International Conference on Cooperative Information Systems (CoopIS 02)*, pages pp. 19 – 35. Springer Verlag LNCS 2519, 2002.
- [DM89] Y. Demazeau and J.-P. Miller. Decentralized Artificial Intelligence. In Y. Demazeau and J.-P. Miller, editors, *Decentralized A.I. – Proc. of the First European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, pages 3–13. North-Holland, 1989.
- [DPS98] Thomas Devogele, C. Parent, and S. Spaccapietra. On spatial database integration. *Int. J. Geographical Information Science*, 12(4):335–352, 1998.

- [GAGCDAFC99] M. Gómez-Albarrán, P.A. Gonzàles-Calero, B. Díaz-Agudo, and C. Fernández-Conde. Modelling the CBR Life Cycle Using Description Logics. In K.-D. Althoff, R. Bergamnn, and L.K. Branting, editors, *Proc. of the International Conference on Case-Based Reasoning, ICCBR'99*, volume 1650 of *Lecture Notes in Artificial Intelligence*, pages 147–161. Springer, 1999.
- [GG01] C. Ghidini and F. Giunchiglia. Local model semantics, or contextual reasoning = locality + compatibility. *Artificial Intelligence*, 127(2):221–259, 2001.
- [GPG⁺99] H.H. Gu, Y. Perl, J. Geller, M. Halper, and M Singh. A methodology for partitioning a vocabulary hierarchy into trees. *Artificial Intelligence in Medicine*, 15:77–98, 1999.
- [GS00] C. Ghidini and L. Serafini. Distributed first order logics. In *Proceedings of the Frontiers of Combining Systems*, pages 121–139, 2000.
- [GW00] N. Guarino and C Welty. Towards a methodology for ontology-based model engineering. In *ECOOP-2000 Workshop on Model Engineering, (Cannes, France)*, 2000.
- [Hal89] T. Halpin. *A logical analysis of information systems: static aspects of the data-oriented perspective*. PhD thesis, University of Queensland, Brisbane, Australia, 1989.
- [Hal97] T. Halpin. An interview- modeling for data and business rules. in: Ross, r. (eds.): Database newsletter. *IEEE Transactions on Data and Knowledge Engineering*, 25(2), 1997.
- [Hal01] T. Halpin. *Information Modeling and Relational Databases, 3rd edition*. Morgan-Kaufmann, 2001.
- [HM03] V. Haarslev and R Moeller. Racer: An OWL reasoning agent for the semantic web. In *Proceedings of the International Workshop on Applications, Products and Services of Web-based Support Systems, in conjunction with the 2003 IEEE/WIC International Conference on Web Intelligence*, pages 91–95, 2003.
- [HST99] I. Horrocks, U. Sattler, and S. Tobies. A description logic with transitive and converse roles, role hierarchies and qualifying number restriction. Technical Report 99-08, Technische Universität Dresden, LTCS, 1999.
- [Jar05] M. Jarrar. *Towards methodological principles for ontology engineering*. PhD thesis, Vrije Universiteit Brussel, Belgium, 2005.

- [JDM02] M. Jarrar, J. Demy, and R. Meersman. On using conceptual data modeling for ontology engineering. *Data Semantics*, 2800:185–207, October 2002. Special issue on Best papers from the ER/ODBASE/COOPIS 2002 Conferences.
- [JM02] M. Jarrar and R. Meersman. Scalability and knowledge reusability in ontology modeling. In *Proceedings of the International conference on Infrastructure for e-Business, e-Education, e-Science, and e-Medicine (SSGRR2002)*, 2002.
- [JVM03] M. Jarrar, R. Verlinden, and R. Meersman. Ontology-based customer complaint management. In M. Jarrar and A. Salaun, editors, *Proceedings of the workshop on regulatory ontologies and the modeling of complaint regulations*, volume 2889 of *LNCS*, pages 594 – 606. Springer Verlag, 2003.
- [KLG98] G. Kamp, S. Lange, and C. Globig. Related Areas. In M. Lenz, B. Bartsch-Sprl, H.-D. Burkhard, and S. Wess, editors, [LBSBW98], volume 1400 of *Lecture Notes in Artificial Intelligence*, chapter 13. Springer, 1998.
- [KLP⁺04] U. Keller, R. Lara, A. Pollares, I. Toma, M. Kifer, and D. Fensel. Wsmo web service discovery. D5.1 v 0.1, wsml working draft, 2004.
- [LBSBW98] M. Lenz, B. Bartsch-Sprl, H.-D. Burkhard, and S. Wess, editors. *Case-Based Reasoning Technology: From Foundations to Applications*, *LNAI 1400*, volume 1400 of *Lecture Notes in Artificial Intelligence*. Springer, 1998.
- [LN98] Jean Lieber and Amedeo Napoli. Correct and Complete Retrieval for Case-Based Problem-Solving. In H. Prade, editor, *Proc. of the European Conference on Artificial Intelligence, ECAI'98*, pages 68–72. John Wiley & Sons Ltd, Chichester, 1998.
- [LRK04] H. Lausen, D. Roman, , and U. Keller. Web service modeling ontology - standard (wsmo-standard), v. 1.0. Working draft, deri. available online <http://www.wsmo.org/2004/d2/v1.0>, 2004.
- [Mee99] R. Meersman. The use of lexicons and other computer-linguistic tools. In Y. Zhang, M. Rusinkiewicz, and Y. Kambayashi, editors, *Semantics, Design and Cooperation of Database Systems, in The International Symposium on Cooperative Database Systems for Advanced Applications (CODAS 99)*, *LNCS*. Springer Verlag, 1999.
- [MI01] Eduardo Mena and Arantza Illarramendi. *Ontology-Based Query Processing for Global Information Systems*. Kluwer, 2001.

- [MLN98] E. Melis, J. Lieber, and A. Napoli. Reformulation in Case-Based Reasoning. In B. Smyth and P. Cunningham, editors, *Proc. of the European Workshop on Case-Based Reasoning, EWCBR'98*, Lecture Notes in Artificial Intelligence 1488, pages 172–183. Springer, 1998.
- [MM01] Brian S. Mitchell and Spiros Mancoridis. Comparing the decompositions produced by software clustering algorithms using similarity measurements. In *Proceedings of the 17th IEEE International Conference on Software Maintenance (ICSM 2001)*, pages 744–753, Florence, Italy, nov 2001.
- [MW95] Mala Mehrotra and Chris Wild. Analyzing knowledge-based systems with multiviewpoint clustering analysis. *Journal of Systems and Software*, 29(3):235 – 249, 1995. Special issue on software quality in knowledge-based systems.
- [NLL96] M.V. Nagendra Prasad, V.R. Lesser, and S.E. Lander. Retrieval and Reasoning in Distributed Case Bases. *Journal of Visual Communication and Image Representation*, 7(1):74–87, 1996.
- [Par72] D.L. Parnas. On the criteria to be used in decomposing system into modules. *Communications of the ACM*, 15(12):1053–1058, December 1972.
- [PSZ05] C. Parent, S. Spaccapietra, and E. Zimanyi. The MurMur project: Modeling and querying multi-representation spatio-temporal databases. *Information Systems*, 2005. To appear. Also available online at <http://www.sciencedirect.com/>.
- [RBG+97] A. Rector, S. Bechhofer, C. Goble, I. Horrocks, W. Nowlan, and W Solomon. The GRAIL concept modelling language for medical terminology. *Artificial Intelligence in Medicine*, 9:139–171, 1997.
- [Rec99] A Rector. Clinical terminology: Why is it so hard? *Methods of Information in Medicine*, 38:239–252, 1999.
- [Rou04] M-C. Rousset. Small can be beautiful. In *Proceedings of the 3rd International Semantic Web Conference (ISWC 2004)*, volume 3298 of LNCS, pages 6–16, 2004.
- [RPR+98] J.E. Rogers, C. Price, A.L. Rector, W.D. Solomon, and N Smejko. Validating clinical terminology structures: Integration and cross-validation of read thesaurus and GALEN. *Journal of the American Medical Informatics Association*, pages 845–849, 1998.
- [RR96] J. Rogers and A Rector. The galen ontology. In *Medical Informatics Europe (MIE 96)*, pages 174–178. IOS Press, 1996.

- [RSB98] C. Rosse, I.G. Shapiro, and J.F. Brinkley. The digital anatomist foundational model: Principles for defining and structuring its concept domain. *Journal of the American Medical Informatics Association*, pages 820–824, 1998.
- [RWRR01] A. Rector, C. Wroe, J. Rogers, and A. Roberts. Untangling taxonomies and relationships: Personal and practical problems in loosely coupled development of large ontologies. In *Proceedings of the First International Conference on Knowledge Capture(K-CAP 2001)*, pages 139–146. ACM, 2001.
- [RZS⁺99] A.L. Rector, P.E. Zanstra, W.D. Solomon, J.E. Rogers, R. Baud, W. Ceusters, J. Claassen Kirby, J.-M. Rodrigues, A.R. Mori, E.v.d. Haring, and J. Wagner. Reconciling users’ needs and formal requirements: Issues in developing a re-usable ontology for medicine. *IEEE Tran on Information Technology in BioMedicine*, 2:229–242, 1999.
- [SJ02] Kiril Simov and Stanislav Jordanov. Bor: a pragmatic daml+oil reasoner. Deliverable 40, On-To-Knowledge Project, June 2002.
- [SK93] A. Sheth and V. Kashyap. So far (schematically) yet so near (semantically). In D.K. Hsiao, E.J. Neuhold, and R. Sacks-Davis, editors, *Interoperable Database Systems (DS-5)*, pages 283–312. IFIP Trans. A-25, North-Holland, 1993.
- [SK03a] H. Stuckenschmidt and M. Klein. Integrity and change in modular ontologies. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI 2003)*, 2003.
- [SK03b] H. Stuckenschmidt and M. Klein. Modularization of ontologies. Deliverable 21. wonderweb project (ist 2001-33052), 2003.
- [SK04a] Heiner Stuckenschmidt and Michel Klein. Ontology refinement — towards structure-based partitioning of large ontologies. Wonderweb Deliverable, 2004.
- [SK04b] Heiner Stuckenschmidt and Michel Klein. Structure-based partitioning of large concept hierarchies. In Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors, *Proceedings of the Third International Semantic Web Conference (ISWC 2004)*, pages 289–303, Hiroshima, Japan, nov 2004.
- [Smy96] B. Smyth. *Case-Based Design*. PhD. thesis, Trinity College, University of Dublin, April 1996.

- [SP94] S. Spaccapietra and C. Parent. View integration: A step forward in solving structural conflicts. *IEEE Transactions on Data and Knowledge Engineering*, 6(2), 1994.
- [ST05] L. Serafini and A. Tamin. Drago: Distributed Reasoning Architecture for the Semantic Web. In A. Gomez-Prez and J. Euzenat, editors, *Proc. of the European Semantic Web Conference, ESWC 2005, LNCS 3532*, pages 361–376. Springer-Verlag, 2005.
- [SWCH01] K. Sullivan, G. William, Y. Cai, and B. Hallen. The structure and value of modularity in software design. *SIGSOFT Software Engineering Notes*, 26(5):99–108, 2001.
- [SWRR99] W. Solomon, C. Wroe, J.E. Rogers, and A Rector. A reference terminology for drugs. *Journal of the American Medical Informatics Association*, pages 152–155, 1999.
- [vBHvdW91] P. van Bommel, A.H.M. Hofstede, and Th.P. van der Weide. Semantics and verification of object role models. *Information Systems*, 16(5):471–495, October 1991.
- [WC01] C. Wroe and J Cimino. Using openGALEN techniques to develop the hl7 drug formulation vocabulary. In *Proceedings of the American Medical Informatics Association Fall Symposium (AMIA-2001)*, pages 766–770, 2001.
- [WSC⁺04] H. Wache, L. Serafini, R. Garcia Castro, P. Groot, M. Jarrar, Y. Kompatsiaris, D. Maynard, J. Pan, F. Roelofsen, S. Spaccapietra, G. Stamou, A. Tamin, and I. Zaihrayeu. Scalability - state of the art. Deliverable d2.1.1, eu-ist network of excellence (noe) ist-2004-507482 (kweb), 2004.

Related deliverables

A number of Knowledge web deliverable are clearly related to this one:

Project	Number	Title and relationship
KW	D2.1.1	D2.1.1 Survey of Scalability Techniques for Reasoning with Ontologies gives an overview of methods for modularisation and distributed reasoning.
KW	D2.2.1	D2.2.1 Specification of a common framework for characterizing alignment discusses also modularisation of large ontologies.
WonderWeb	D2.1.1	D21 Modularization of Ontologies discusses the infrastructure and some aspects of modularisation.