Proceedings of the ECAI-02 Workshop on Ontologies and Semantic Interoperability

J. Euzenat

A. Gómez Pérez

N. Guarino

H. Stuckenschmidt

(*editors*)

Held on July 22nd, 2002 in conjunction with the

European Conference on Artificial Intelligence

Lyon, France

ECAI-02 Workshop on
Ontologies and Semantic Interoperability

Held on July 22nd, 2002 in conjunction with the

European Conference on Artificial Intelligence

Lyon, France

The ECAI-02 Workshop on Ontologies and Semantic Interoperbability is a follow up of a series of successful Workshops on Ontologies and related topics that have been held at major AI and Computer Science Conferences over the last years. With the increasing interest in the Semantic Web Ontologies have become a major topic in many conferences and workshops. On the one hand, this leads to a wider acceptance and use of ontologies, on the other hand the field is in danger of of loosing focus. This workshop is meant to provide a forum for reserachers interested in Ontologies as a core technology for intrelligent information processing, thereby trying to shape the area and its specific topics. At the IJCAI-01 Workshop on Ontologies an Information Sharing, there has been an agreement that interoperability will be a main issue in reserach over the next years. Consequently, this workshop, besides covering general topics connected to ontologies, puts a special emphasis on Semantic Interoperability.

We received 14 submissions of which 6 were accepted as technical papers. The programme of the workshop is supplemented by 4 papers describing existing systems and implementations of infrastructure supporting the use of ontologies.

Technical Papers

The six technical papers that have been selected by the programme committee cover a wide range of topic connected with ontology technology and ist use:

Menzel (page 38) reports first attempts of developing a better theory of ontological knowldge by investigating requirements of an ontology theory and comparing them with the formal properties of Common Logic.

Fernandez-Lopez (page 6) as well as Gangemi et.al. (page 16) address the problem of building ontologies. While the former paper discusses the problem at a general level, proposing to use a metamodel of the creation process to explicate modeling bias, the latter paper reports on a concrete attempt to build a well founded ontology for a specific domain by combining and extending existing sources of ontological information.

The work by Mitra and Wiederhold (page 45) as well as by Tamma and Bech-Capon (page 51) is concerned with the problem of re-using ontologies, however focussing on different scenarios. Mitra and Wiederhold present different Methods for resolving termoinological heterogeneity between pre-existing ontologies. Tamma and Bech-Capon propose an extension of the OntoClean Framework for ontological analysis that focusses on the meta-properties of class attributes and show how this extension supports the comparison of different ontologies

Finally, Klein and others (page 31) discuss the special problem of managing different versions of the same ontology. Building upon earlier work they discuss different kinds of changes in aan ontology and describe how versions of an ontology can be compared.

System Descriptions

The four system descriptions accepted for presentation at the workshop provide an idea of the variety of assisting tools needed for and applications supported by ontology technology.

Fiedler and others (page 62) present Ωmega, a reasoning system that supports the representation and explanation of proofs focussing on the role of ontologies in providing this functionality.

Hammer and others (pages 67) describe a system that supports the integration of different legacy information systems by automatically extracting conceptual models from databases.

Visser and Schuster (page 74) describe the use of ontologies for finding and combining simple web services in terms of information providers using ontology-based metadata descriptions.

Volz and Maedche, finally (page 80), describe a computational infrastructure for supporting the use of modularized ontologies focussing on different import mechanisms and their implementation.

Program Committee and Reviewers.

We are grateful to the following members of the international program committee for helping us to make this a high quality workshop:

- Joost Breuker, University of Amsterdam, NL
- Mike Brown, SemanticEdge, D
- Oscar Corcho, Politecnical University Madrid, ES
- Rose Dieng, INRIA Sophia Antipolis, F
- Jérôme Euzenat, INRIA Rhones-Alpes, F
- Fausto Giunchiglia, University of Trento, I
- Asunción Gómez-Pérez, Politecnical University Madrid, ES
- Michael Gruninger, NIST, USA
- Nicola Guarino, LADSEB-CNR, Padova, I
- Michel Klein, Free University Amsterdam, NL
- Alexander Maedche, FZI Karlsruhe, D
- Chris Menzel, Texas A&M University, USA
- Enrico Motta, Open University, UK
- Leo Obrst, Mitre Corporation, USA
- Guus Schreiber, University of Amsterdam, NL
- Steffen Staab, University of Karlsruhe, D
- Heiner Stuckenschmidt, University of Bremen, D
- Rudi Studer, AifB, University of Karlsruhe, D
- Gerd Stumme, University of Karlsruhe, D
- Valentina Tamma, University of Liverpool, UK
- Mike Uschold, Boeing Corporation, USA
- Ubbo Visser, University of Bremen, D

We would also like to thank the following additional reviewers:

- Stefano Borgo
- Bob Colomb
- Raphael Troncy

Table of Contents

Technical Papers

Meta-modelling for ontology development and knowledge exchange

Mariano Fernández-López
Laboratorio de Inteligencia Artificial
Facultad de Informática
Universidad Politécnica de Madrid
Campus de Montegancedo sn.
Boadilla del Monte, 28660. Madrid, Spain.
Tel: (34-91) {336-66-05, 336-74-39}
Fax: (34-91) 3-52-48-19
Email: mfernandez@fi.upm.es

ABSTRACT

One of the sources of heterogeneity of ontologies is that different ontologies have different necessities of modelling. This paper presents a bi-phase method to deal with these different necessities. Phase I of the method models how to model the ontology, obtaining a meta-model. Such meta-model can be expressed in LBIR, a formal and declarative language that has been specifically designed for this task. To save resources, a reference meta-model that can be modified and reused is provided. During phase II of the method, the ontology is modelled following the meta-model obtained in the first phase. Furthermore, a tool (called ODE) provides software support to the method. Such tool generates SQL schemas from LBIR, and allows the modelling of the ontology following the selected meta-model. This approach eases the interoperability between groups located in different geographical locations that have to build the same ontology, since the meta-model to be used can be exchanged through LBIR.

KEYWORDS

Ontology, meta-model, modelling, method, LBIR, ODE.

1. EXPOSITION OF THE PROBLEM

Even though Ontological Engineering is a very young area in Artificial Intelligence, there exist some methodological proposals for building ontologies: Uschold and King's methodology [Usc95], Grüninger and Fox's methodology [Grü95], METHONTOLOGY [FeG99], etc. A study and analysis of methodologies for building ontologies can be found at [Fer99]. This study shows that METHONTOLOGY is currently the most mature methodology.

Presently, methodologies do not propose to adapt the mechanism of modelling to the different ontologies to be built. However, our experience in different projects (the (KA)^2 initiative [Ben98], the multidisciplinary project AM9819 about environmental pollutants, etc.) show that different domains should be modelled in different ways. Table 1 shows the components that have been used in different ontologies. We can see that there are variations from some ontologies to others. Some ontologies have been built using a lot of attributes and no relations, others have been built using constants, some of them have first order logic formulas, but others do not, etc.

Apparently, one solution to this problem would be to consider all the "necessary" components (concepts, attributes, first order logic formulas, constants, etc.) when an ontology had to be built. Nevertheless, such solution has the following drawbacks: (1) Our experience has shown it is possible that need for a component is not perceived a priori, that is, it is possible the necessity of a component is only detected when it is needed in an ontology. (2) New research about modelling can provide new components and new ideas about how to use old components. (3) Considering non-useful components when an ontology is built can cause confusion in modellers, and especially when they are not very experienced.

Besides flexibility in the components to be used during the modelling, the knowledge should be presented in different ways to different experts.

Summarising, **a rigid way to model brings us back to the classic knowledge-acquisition bottleneck** [Eri99].

Ontology	Domain	Concepts	Instance attributes	Relations	Constants	First order logic formulas	Arithmetic formulas	Instances	TOTAL TERMS
CHEMICALS.1	Chemical	10	6	0	0	0	1	20	37
CHEMICALS.2	Chemical	16	22	0	0	27	3	103	173
CHEMICALS.3	Chemical	16	20	0	2	27	1	103	169
$(KA)^2$ restructured	Knowledge acquisition community	78	12	47	0	0	0	102	239
Reference Ontology	Ontologies	23	70	9	0	0	0	8	110
Standard Units restructured	Measure units	22	3	0	2	0	1	65	93
Monatomic ions	Environmental ions	62	11	3	0	6	0	0	82
Silicates	Silicates	84	17	8	0	0	0	0	109
Hardware	Laboratory of Artificial Intelligence's hardware	49	56	0	0	0	0	56	190
ELLOS Ontology	Catalogue of clothes	8	16	6	0	0	0	20	48
Tradezone Ontology	Catalogue of products in general	9	3	3	0	4	0	0	22
SNCF Ontology	Travels and hotels	13	37	4	4	2	2	1	69
FIDAL Ontology	Contracts	6	15	8	0	1	0	7	37

Table 1. Components used in the ontologies developed with the bi-phase method

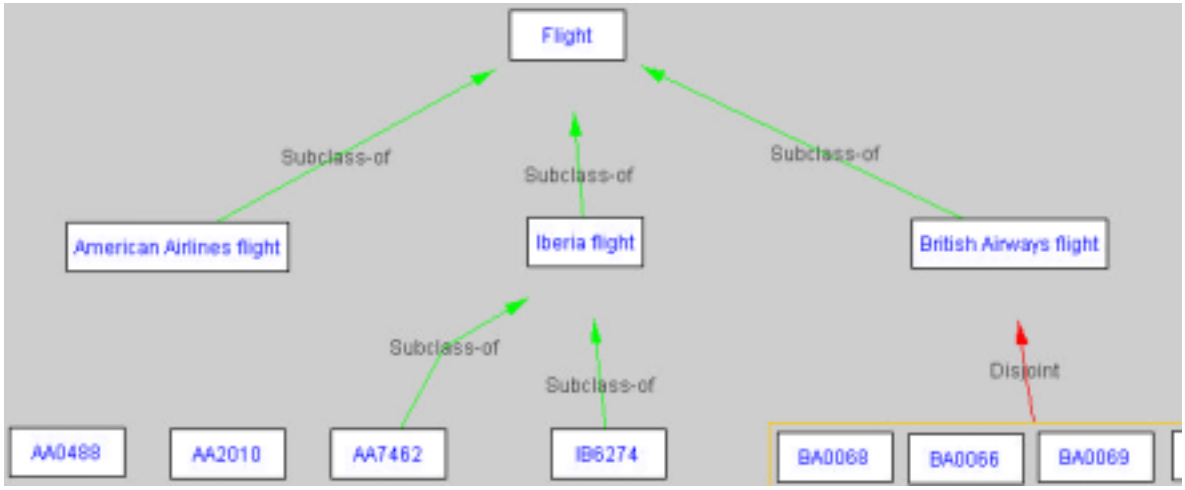

Figure 1. `Concept classification tree` *in the domain of flights*

Concept name	Instances	Instance attributes	Relations
AA0488	--	--	--
AA2010	--	--	--
AA7462	AA7462_Feb08_2002 AA7462_Feb16_2002	--	same flight as
American Airlines flight	--	--	--
BA0066	--	--	--
BA0067	--	--	--
BA0068	--	--	--
BA0069	--	--	--
British Airways flight	--	--	--
Business trip	--	budget	--

Table 2. `Concept dictionary` *in the domain of flights*

Focussing on the case of METHONTOLOGY, it proposes to carry out the following steps to develop an ontology: *specification* in natural language, *conceptualisation* using tables and graphs, *formalisation* (e.g. using frames), and *implementation* (e.g. using the Ontolingua language [Far97]). According to the METHONTOLOGY viewpoint, conceptualisation is the modelling at the knowledge level [New82], hence, the knowledge is modelled independently of the implementation language to be used[1]. The proposed tables and graphs allow modelling concepts, attributes, first order logic formulas, etc., and they are thought to be manipulated by experts in the domains to be modelled. Figure 1 presents an example of a graph: a `concept classification tree`, and table 2 is an example of `concept dictionary`. Tables and graphs in METHONTOLOGY are not fixed, since the engineer can use tables or graphs that can be different to the proposed ones by the methodology. However, METHONTOLOGY does not propose a precise way to specify how the tables and the graphs to be used during the conceptualisation are. Besides, this methodology does not propose how to add a new type of table, how to add a new field to a type of table, how to

[1] Such idea of conceptualisation is inspired in the Hayes-Roth and colleagues' approach [Hay83].

delete one of the types of the proposed graphs, or how to elaborate a completely new modelling way with completely new graphs and tables. Therefore, if several groups in different locations have to build an ontology collaboratively, **there are problems to agree and exchange the characteristics of the tables and graphs to be used** (see figure 2).

In the following sections, a solution to these problems will be presented. Section 2.1 will present the bi-phase method and, section 2.2, its software support: ODE. The paper will finish with the conclusions and future trends.

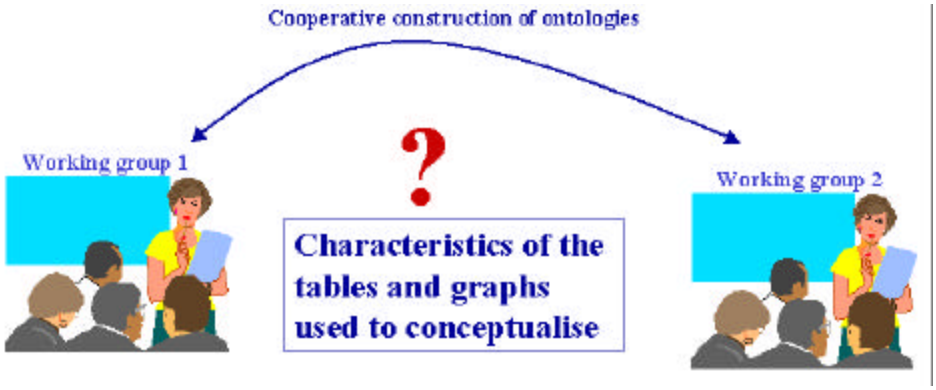

Figure 2. Problems in collaborative construction when the characteristics of tables and graphs are not clearly specified

2. THE PROPOSED SOLUTION

2.1. THE METHODOLOGICAL LEVEL OF THE BI-PHASE SOLUTION

To allow a more flexible modelling of ontologies and to ease the exchange of characteristics of tables and graphs, the **bi-phase method** proposes to model how to model the ontology. Until now, the purpose of the ontology engineer was to model some parts of the world, for example, flights, chemical elements, etc. (see figure 3), however, with the bi-phase method, modelling the process of modelling is also recommended, that is, building a **meta-model** is also proposed. Particularly, the part of the modelling process to model is the conceptualisation, which is the base of the remainder steps of the modelling.

The bi-phase method follows the METHONTOLOGY approach, although in two levels. On the one hand, during phase I, the ontology conceptualisation process is specified in natural language, conceptualised using tables and graphs (called in this phase **meta-tables** and **meta-graphs**), formalised using a formal language, and implemented in SQL (see figure 4). Thus, the result of this first phase is a meta-model presented in meta-tables and meta-graphs, in a formal language, and in SQL. The steps of this phase are called: **meta-specification**, **meta-conceptualisation**, **meta-formalisation** and **meta-implementation**. On the other hand, phase II carries out the specification, conceptualisation (following the meta-model obtained in phase I), formalisation and implementation of the ontology. As you can see, in this bi-phase method, there is a modelling both at Newell's knowledge level and symbolic level during phase I as well as phase II.

To facilitate the building of meta-models, a **reference meta-model** is proposed. It is possible to modify this reference meta-model according to the modelling needs of each ontology. Such meta-model is expressed by means of meta-tables and meta-graphs, and it is also formally expressed. The reference meta-model allows building ontologies with: concepts, class and instance attributes, facets of such attributes, relations, first order logic formulas, arithmetic formulas, constants, and instances. These components appear in the reference meta-model because each one of them have been used in some of the ontologies developed during the experimentation. Besides, we have checked that the reference meta-model contains the static components of the classic languages for ontology development (Ontolingua, OKBC, OCML, FLogic and LOOM). We say static components because we do not consider rules and procedures. This reminds as future work.

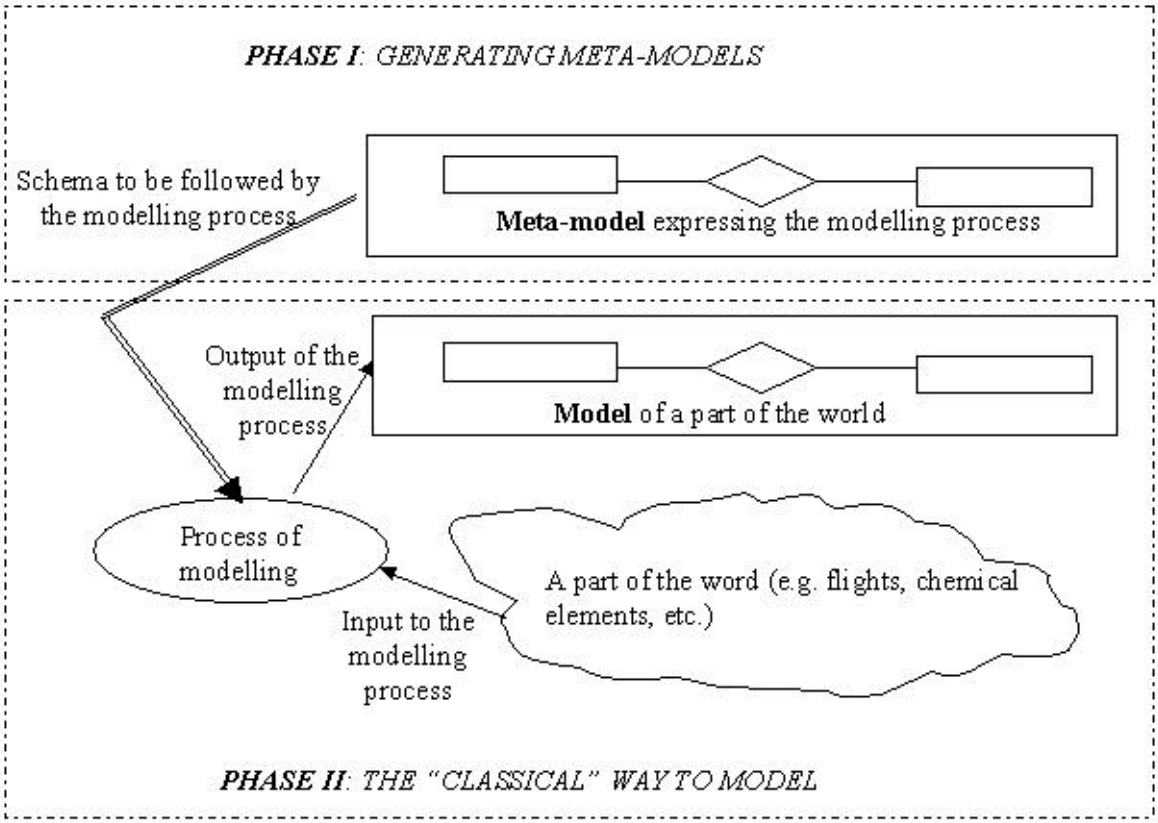

Figure 3. General overview of the ontology development using meta-models

We have also developed a tool, called ODE, in order to provide software support to the bi-phase method. Ode is especially designed to facilitate the application of the method.

However, it is not the only tool allowing flexible modelling, since Protégé-2000 [Fri00] permits the user to redefine its components (made by classes, slots, etc.).

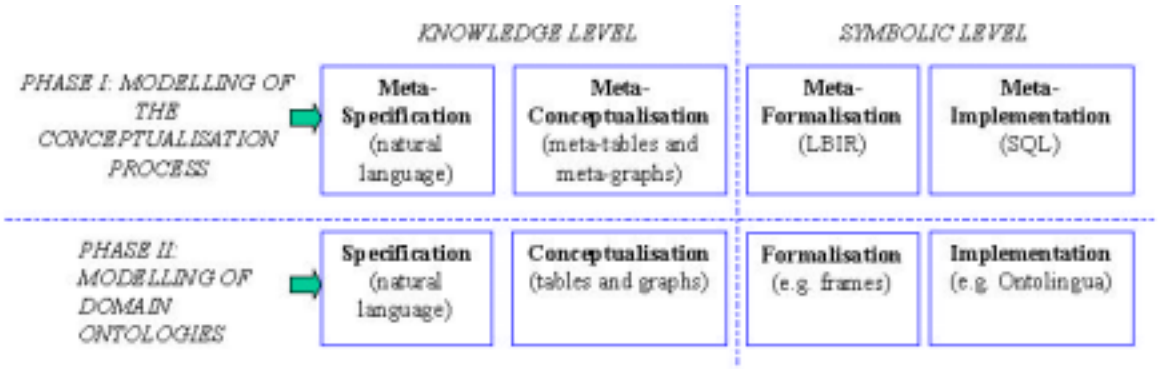

Figure 4. Bi-phase method to build ontologies

In [Fer01] a complete description of the method is presented. Such description includes the tasks to

be performed, the inputs, the outputs and the participants. This description includes a way to manage the changes in meta-models, even when an ontology is being developed with such meta-model and new necessities are detected. There is also a description of the architecture of ODE. The method and the tool have been tested in the above mentioned projects (the (KA)[2] initiative, the multidisciplinary project AM9819 about environmental pollutants, etc.). 10 different meta-models have been built with a total of 33 additions, removals and modifications with regards the reference meta-model; such meta-models have been used in 11 different domains: chemical elements (169 terms with 27 first order formulas), knowledge acquisition community (239 terms with no first order formulas), hardware (190 terms with no first order formulas), ontologies (110 terms with no first order formulas), measure units (93 terms with no first order formulas), monatomic ions (82 terms with 6 first order formulas), silicates (109 terms with no first order formulas), catalogues of cloths (48 terms with no first order formulas), travels (22 terms with no first order formulas), hotels (69 terms with 2 first order formulas) and contracts (37 terms with 1 first order formula). Other meta-models have been built containing meta-graphs and meta-tables to model databases, other meta-models contain meta-graphs to model tasks, and other meta-models even contain schemas of bills, invoices, etc. as meta-tables.

In the following sub-sections, a brief description of the steps of phase I will be presented.

2.1.1. Meta-specification of the conceptualisation process

During phase I, the meta-specification describes, in natural language: (a) what *tables and graphs* will be used during the conceptualisation of the ontology; (b) the *recommended order* to fill in the tables and to build the graphs; and (c) the *consistency verification rules* between tables, between graphs, and between tables and graphs. For example, it can be (meta-)specified that a graph to be used during the conceptualisation is the `concept classification tree`, that the nodes of this graph are `concepts`, and that the edges are `subclass of`, `subclass in a disjoint partition`[2], and `subclass in an exhaustive partition`[3]. Besides, it can be also specified that a table to use during the conceptualisation of the ontology is the `concept dictionary`. The possible fields of such table would be: `concept name`, `instances`, `instance attributes`, etc. Concerning the recommended order, it should be said that the elaboration of the concept classification tree should begin before starting the concept dictionary. And with regard to the consistency verification rules between the concept classification tree and the concept dictionary, all the concepts of the tree should be in the concept dictionary and vice versa.

2.1.2. Meta-conceptualisation of the conceptualisation process

For (meta-)conceptualising in phase I, the bi-phase method proposes: (a) a set of meta-tables to describe the tables and graphs to be used during the conceptualisation in phase II; (b) a meta-graph to describe the order in the conceptualisation in phase II; (c) and meta-tables and meta-graphs to describe the consistency verification rules. Thus, for example, the `meta-tables of node description`, and the `meta-tables of edge description` are proposed to define the details of the graphs, and the `meta-tables of field description` are proposed to define the details of the tables. For instance, meta-tables 1, 2 and 3 show the description of the taxonomy and of the concept dictionary, used both in the examples of section 1.1.1. In all these meta-tables, the meta-field `symbol` is filled in with abbreviations. In the case of meta-table 2, which describes a graph, the meta-fields `input and output edges`, `input multiplicities` and `output multiplicities` are used to establish how many edges can go in and go out to and from a node. In the case of meta-table 3, which describes the concept dictionary, the meta-field `format` restricts the possibilities to fill in the cells (text, list, logic expression, etc). `Is it main` is true when the described field is the identifier of the row. `Repetition in the same table` is true when the field can be filled in with the same value in different rows. And `multiplicity` is true when the same cell can have several values.

[2] 'Subclass in a disjoint partition'. A disjoint partition of a class is a set of subclasses of this class that do not have common instances.

[3] 'Subclass in an exhaustive partition'. An exhaustive partition of a class is a set of subclasses that covers all the class, that is, there is not an instance of the father class that is not an instance of any of the subclasses of the partition.

Edge	Symbol	Description
Subclass of	S	A class C is a subclass of the parent class P if and only if every instance of C is also an instance of P.
Subclass in a disjoint partition	SDP	A disjoint partition of a class is a set of its subclasses where the subclasses do not have common instances.
Subclase in an exhaustive partition	SEP	An exhaustive partition of a class is a set of subclasses that covers all the class, that is, there is no instance of the father subclass that is not subclass of any class of the subclasses of the partition

Meta-table 1. `Meta-table of edge description` *defining the possible edges of the graph "concept classification tree"*

Node	Symbol	Descrip-tion	Input and outpud edges	Input multipli-cities	Output multipli-cities
Concept	C	**	Subclass of	(0, n)	(0, n)
			Subclass in a disjoint partition	(0, n)	(0, n)
			Subclass in an exhaus-tive parti-tion	(0, n)	(0, n)

Meta-table 2. `Meta-table of node description` *defining the possible nodes of the graph "concept classification tree"*

The meta-graph to model the order during the conceptualisation is not presented due to the space constraints. Concerning the consistency verification rules between tables, between graphs, and between tables and graphs, the way to write them is based on operations on matrices representing the tables and the graphs. Such operations are similar to the ones used in the relational model for databases (projection, selection, difference, etc.).

2.1.3. Meta-formalisation and meta-implementation of the conceptualisation process

To carry out the meta-formalisation, a formal and declarative language, called **LBIR** (Language for Building Intermediate Representations), has been elaborated. Such language has the same expressiveness as the meta-tables and meta-graphs used during the meta-conceptualisation. The LBIR description uses a context free grammar for the syntax, and matrices to establish the meaning of the language. The following code:

```
define table horizontal [Concept dictionary] as CD

define field [Concept name] as CN
        begin
          type term;
          repeated no;
          multiplicity (1,1);
        end field;
define field Instances as I
        begin
          type term;
          repeated yes;
          multiplicity (0,N);
define field [Instance attributes] as IA
        begin
          type term;
          repeated yes;
          multiplicity (0,N);
        end field;
define field Relations as R
        begin
          type term;
          repeated yes;
          multiplicity (0,N);
        end field;
begin
        placed in [Binary relation diagram];
        main field [Concept name];
end table;
```

shows the definition in LBIR of the concept dictionary, that is equivalent to the definition appearing in meta-table 3. `Placed in binary relation diagram` indicates that a graph called binary relation diagram should be designed before filling in he concept dictionary.

Field	Symbol	Description	Format	Is it main?	Repetition in the same table	Multiplicity
Concept name	CN	**	Term	Yes	No	(1, 1)
Instances	I	Instances are particular cases of the concept	Term	No	Yes	(0, n)
Instance attributes	IA	The ones that allow describing the instances of the concept.	Term	No	Yes	(0, n)
Relations	R	Relations link concepts	Term	No	Yes	(0, n)

Meta-table 3. `Meta-table of field description` *defining the table "concept dictionary"*

During the meta-implementation, the meta-model expressed in LBIR is transformed into a SQL schema. This eases the use of *databases to store ontologies*, taking advantage of the independence and integrity of the data, the minimisation of the

redundancy, etc., provided by the relational database systems.

3.2. THE SOFTWARE LEVEL OF THE BI-PHASE SOLUTION

In order to allow the efficient use of the methodology proposed in 3.1, we has built ODE (see figure 5). The *LBIR processing module* automates the transformation, without loss of expressiveness, from a meta-model in LBIR to a SQL schema. Besides, it allows conceptualising ontologies following a meta-model selected by the user, and storing the result in a database following the SQL schema associated to the meta-model. Moreover, if you follow the reference meta-model to conceptualise your ontology you can use a generator of Ontolingua code. The main feature of ODE is that a change in the meta-model does not force a change in the program, since SQL schemas are generated in run-time and not in design time (as usual).

3. CONCLUSIONS AND FUTURE TRENDS

Although each ontology has its modelling needs, there is not any methodological proposal to use a different kind of modelling for each ontology.

The bi-phase method presented in this paper proposes, during a first phase, to model the modelling process itself (or reusing an existing meta-model) and, during the second phase, to model the ontology. In the first phase, the steps are: meta-specification, meta-conceptualisation, meta-formalisation and meta-implementation. During the second phase the steps are the ones proposed by METHONTOLOGY: specification, conceptualisation, formalisation and implementation. To carry out the meta-formalisation, a formal and **declarative language** (LBIR) has been elaborated. Moreover, to provide software support to both phases, a tool has been developed: ODE.

To agree in the meta-model to be used, the different groups can exchange this meta-model in meta-tables and meta-graphs, or in LBIR (see figure 6). The second option, LBIR, is mandatory if the current version of ODE is utilised to build the ontologies.

The method and the tool have proved useful in several Spanish and international projects.

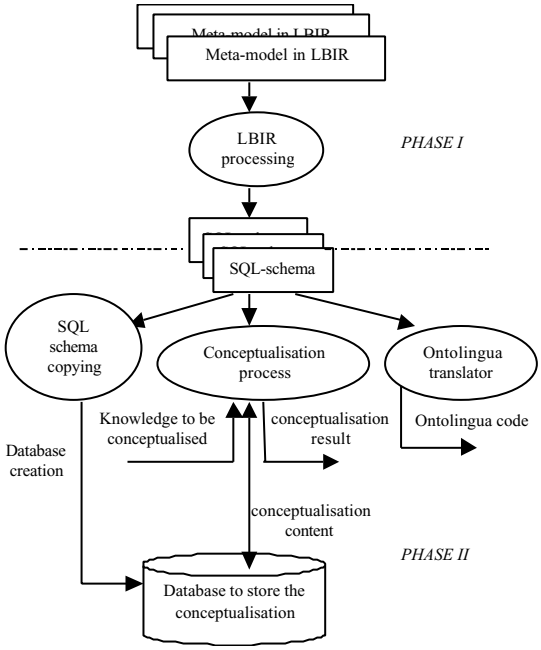

Figure 5. ODE processes

One of the most interesting future lines, above all for ODE, would be the fast development of translators from different meta-models into different implementation languages. An interface to manipulate meta-tables and meta-grpahs would be also interesting. Another important future trend would be a structured characterisation of ontologies according to their modelling needs. Now, the modelling necessities are determined by the experience of the ontology engineers, who interacts with the experts in the domain to be modelled.

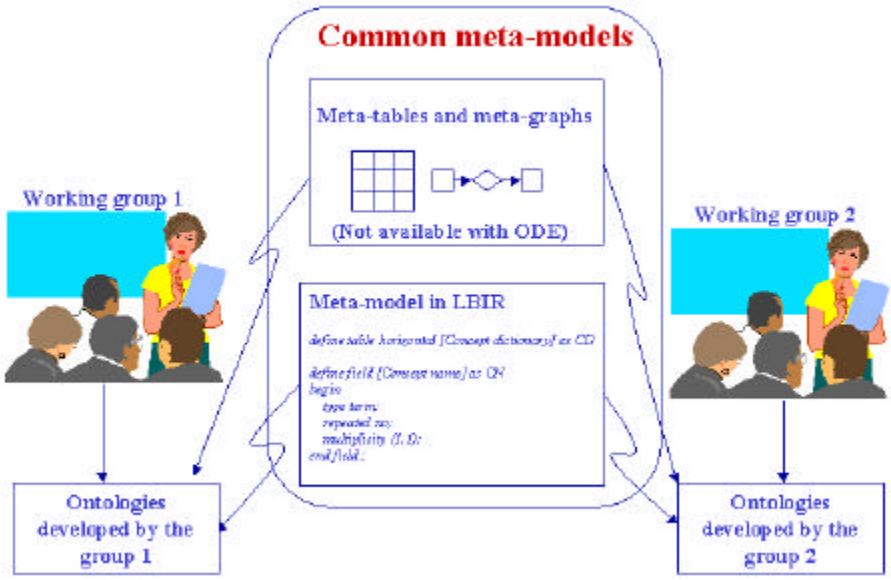

Figure 6. Use of meta-models for cooperative construction of ontologies

4. REFERENCES

[Ben98] Benjamins, V.R.; Fensel, D.; Gómez-Pérez, A.; Decker, S.; Erdmann, M.; Motta, E.; Musen, M. 1998. "The Knowledge Annotation Initiative of the Knowledge Acquisition Community (KA)^2". In: *Proceedings of the 11th Banff Knowledge Acquisition for Knowledge-Based System Workshop (KAW 98)*, Banff, Canada.

[Eri99] Eriksson, H.; Fergerson, R.W.; Shahar, Y.; Musen, M.A. "Automatic generation of ontology editors". Knowledge Acquisition Workshop (KAW). Banff (Canada). 1999.

[Far97] Farquhar, A.; Fikes, R.; Rice, J. "The Ontolingua Server: Tool for Collaborative Ontology Construction". IJHCS, 46(6) (1997) 707-728.

[Fen00] Fensel, D. *Ontologies: silver bullet for Knowledge Management and Electronic Commerce*. Springer-Verlag. Berlin. 2000.

[Fer01] Fernández-López, M. *Método bi-fase para la conceptualización de ontologías basado en meta-modelos.* Ph. Thesis. Facultad de Informática. Universidad Politécnica de Madrid. Spain. 2001.

[FeG99] Fernández-López, M.; Gómez-Pérez, A.; Pazos-Sierra, A.; Pazos-Sierra, J. "Building a Chemical Ontology Using METHONTOLOGY and the Ontology Design Environment". *IEEE Intelligent Systems & their applications.* January/February. Pages 37-46. 1999.

[Fer99] Fernández López, M. "Overview of Methodologies for Building Ontologies". *Workshop on Ontologies and Problem-Solving Methods: Lessons Learned and Future Trends. (IJCAI99)*. August. 1999.

[Fri00] Fridman Noy, N.; Fergeson, R.W.; Musen, M.A. "The knowledge model of Protégé-2000: combining interoperability and flexibility". *Workshop on Knowledge Acquisition, Modelling and Management (EKAW)*. Editor Springer Verlag. Jean Les Pins (Francia). 2000.

[Grü95] Grüninger, M.; Fox, M. S. "Methodology for the design and evaluation of ontologies". *Workshop on Basic Ontological Issues in Knowledge Sharing. Montreal*, Canada. 1995.

[Hay83] Hayes-Roth, F.; Waterman, D. A.; Lenat, D. B. "Building Expert Systems". *Addison Wesley Publishing Company, Inc*. Massachusets, USA. 1983.

[Nec91] Neches, N.; Fikes, R.; Finin, T.; Gruber, T.; Patil, R.; Senator, T.;

Swartout, W.R. “Enabling technology for knowledge sharing”. *AI Magazine, Fall* 1991. Pages 36-56. 1991.

[New82] Newell, A. “The Knowledge Level”. Artificial Intelligence. Volume 18. Number 1. Pp. 87-127. January 1982.

[Usc95] Uschold, M. King, M. “Towards a methodology for building ontologies”. *Workshop on Basic Ontological Issues in Knowledge Sharing*. *International Joint Conference on Artificial Intelligence* (IJCAI). Montreal, Canada. 1995.

A Formal Ontological Framework for Semantic Interoperability in the Fishery Domain

Aldo Gangemi[1], Frehiwot Fisseha[2], Ian Pettman[3], Domenico M. Pisanelli[1], Marc Taconet[4], Johannes Keizer[2]

[1] Institute of Psychology, CNR (National Research Council), Rome, Italy
{gangemi,pisanelli}@ip.rm.cnr.it
http://saussure.irmkant.rm.cnr.it
[2] FAO-GILW, Rome, Italy
{Frehiwot.Fisseha,Johannes.Keizer}@fao.org
http://www.fao.org
[3] One Fish, SIFAR, Grange-over-Sands, Cumbria, UK
ip@ceh.ac.uk
http://www.onefish.org[3]
[4] FIDI, FAO, Rome, Italy
Marc.Taconet@fao.org
http://www.fao.org

Abstract. This paper outlines a project (involving FAO, SIFAR, and CNR) aimed at building an ontology in the fishery domain. The ontology will support semantic interoperability among existing fishery information systems and will enhance information extraction and text marking, envisaging a fishery semantic web. The ontology is being built through the conceptual integration and merging of existing fishery terminologies, thesauri, reference tables, and topic trees. Integration and merging are shown to benefit from the methods and tools of formal ontology.

1 INTRODUCTION

1.1 The general problem

Specialized distributed systems are the reality of today's information systems architecture. Developing specialized information systems/resources in response to specific user needs and/or area of specialization has its own advantage in fulfilling the information needs of target users. However, such systems usually use different knowledge organization tools such as vocabularies, taxonomies and classification systems to manage and organize information. Although the practice of using knowledge organization tools to support document tagging (thesaurus-based indexing) and information retrieval (thesaurus-based search) improves the functions of a particular information system, it is leading to the problem of integrating information from different sources due to lack of semantic interoperability that exists among knowledge organization tools used in different information systems.

The different fishery information systems and portals that provide access to fishery information resources are one example of such scenario. This paper demonstrates the proposed solution to solve the problem of information integration in fishery information systems. The proposal shows how a fishery ontology that integrates the different thesauri and taxonomies in the fishery domain could help in integrating information from different sources be it for a simple one-access portal or a sophisticated web services application.

1.2 The local scenario

Fishery Ontology Service (FOS) is a key feature of the Enhanced Online Multilingual Fishery Thesaurus, a project aimed at information integration in the fishery domain. It undertakes the problem of accessing and/or integrating fishery information that is already partly accessible from dedicated portals and other web services.

The organisations involved in the project are: FAO Fisheries Department (FIGIS), ASFA Secretariat, FAO WAICENT (GIL), the oneFish service of SIFAR, and the Ontology and Conceptual Modelling Group at ISTC-CNR. The systems to be integrated are: the "reference tables" underlying the FIGIS portal [1], the ASFA online thesaurus [2], the fishery part of the AGROVOC online thesaurus [3], and the oneFish community directory [4].

The official task of the project is "to achieve better indexing and retrieval of information, and increased interaction and knowledge sharing within the fishery community". The focus is therefore on tasks (indexing, retrieval, and sharing of mainly documentary resources) that involve recognising an *internal structure* in the content of texts (documents, web sites, etc.). Within the semantic web community and the intelligent information integration research area (cf. [5] and [6]), it is becoming widely accepted that content capturing, integration, and management require the development of detailed, formal *ontologies*.

In this paper we sketch an outline of the FOS development and some hint of the functionalities that it carries out.

2 ONTOLOGY INTEGRATION AND MERGING

2.1 Heterogeneous systems give heterogenous interpretations

An example of how formal ontologies can be relevant for fishery information services is shown by the information that someone could get if interested in *aquaculture*.

In fact, beyond simple keyword-based searching, searches based on tagged content or sophisticated natural-language techniques require some conceptual structuring of the linguistic content of texts. The four systems concerned by this project provide this structure in very different ways and with different conceptual

'textures'. For example, the AGROVOC and ASFA thesauri put *aquaculture* in the context of different thesaurus hierarchies; an excerpt of the AGROVOC result is (*uf* means *used for*, *NT* means *narrower than*; *rt* means *related term, Fr* and *Es* are the corresponding French and Spanish terms):

AQUACULTURE
uf aquiculture
uf mariculture
uf sea ranching
NT1 fish culture
NT2 fish feeding
NT1 frog culture

rt agripisciculture
rt aquaculture equipment

Fr aquaculture
Es acuicultura

The AGROVOC thesaurus seems to frame aquaculture from the viewpoint of *techniques* and *species*. On the other hand, the ASFA aquaculture hierarchy is substantially different:

AQUACULTURE
uf Aquaculture industry
uf Aquatic agriculture
uf Aquiculture
NT Brackishwater aquaculture
NT Freshwater aquaculture
NT Marine aquaculture
rt Aquaculture development
rt Aquaculture economics
rt Aquaculture engineering
rt Aquaculture facilities

Actually this hierarchy seems to stress the *environment* and *disciplines* related to aquaculture.

A different resource is constituted by the so-called *reference tables* in FIGIS system; the only reference table mentioning *aquaculture* puts it into another context (taxonomical species):

Biological entity
Taxonomic entity
Major group
Order
Family
Genus
Species
Capture species (filter)
Aquaculture species (filter)
Production species (filter)

Tuna atlas spec

The last resource examined is oneFish directory, which returns the following context (related to *economics* and *planning*):

SUBJECT
Aquaculture
Aquaculture development
Aquaculture economics @
Aquaculture planning

With such different interpretations of *aquaculture*, we can reasonably expect different search and indexing results. Nevertheless, our approach to information integration and ontology building is not that of creating a homogeneous system in the sense of a reduced freedom of interpretation, but in the sense of navigating alternative interpretations, querying alternative systems, and conceiving alternative contexts of use.

To do this, we require a comprehensive set of ontologies that are designed in a way that admits the existence of many possible pathways among concepts under a common conceptual framework. This framework should reuse domain-independent components, be flexible enough, and be focused on the main reasoning schemas for the domain at hand.

Domain-independent, *upper* ontologies should characterise all the general notions needed to talk about economics, biological species, fish production techniques; for example: *parts*, *agents*, *attribute*, *aggregates*, *activities*, *plans*, *devices*, *species*, *regions of space or time*, etc. While the so-called *core* ontologies should characterise the main conceptual habits (schemas) that fishery people actually use, namely that certain plans govern certain activities involving certain devices applied to the capturing or production of a certain fish species in certain areas of water regions, etc.

Upper and core ontologies [7,8] provide the framework to integrate in a meaningful and *intersubjective* way different views on the same domain, such as those represented by the queries that can be done to an information system.

2.2 Methods applied to develop the integrated fishery ontology

Once made clear that different fishery information systems provide different views on the domain, we directly enter the paradigm of *ontology integration*, namely the integration of schemas that are arbitrary logical theories, and hence can have multiple models (as opposed to database schemas that have only one model) [9]. As a matter of fact, thesauri, topic trees and reference tables used in the systems to be integrated could be considered as *informal* schemas conceived to query semi-formal or informal databases such as texts and tagged documents.

In order to benefit from the ontology integration framework, we must transform informal schemas into *formal* ones. In other words, thesauri and other terminology management resources must be transformed into (formal) ontologies.

To perform this task, we apply the techniques of three methodologies: OntoClean [8], ONIONS [10], and OnTopic [11].

The first one contains principles for building and using upper ontologies for core and domain ontology analysis, revision, and development. In its current form, OntoClean also features an axiomatised domain-independent top-level of formal criteria, concepts and relations (Figure 3) [18].

ONIONS is a set of methods for enhancing the informal data of terminological resources to the status of formal ontological data types. Some methods are aimed at reusing the structure of hierarchies (e.g., BT/NT relations, subtopic relation, etc.), the additional relations that can be found (e.g., RT relations), and at analysing the compositional structure of terms in order to capture new relations and definitional elements. Other methods concern the management of semantic mismatches between alternative or overlapping ontologies, and the exploitation of systematic polysemy to discover relevant domain conceptual structures.

OnTopic is about creating dependencies between topic hierarchies and ontologies. It contains methods for deriving the elements of an ontology that describe a given topic, and methods to build 'active'topics that are defined according to the dependency of any individual, concept, or relation in an ontology.

In Figure 1, a class diagram is shown of the informal and formal data types taken into account by the forementioned methodologies.

In section 3.1 the types of (meta)data extracted from the resources are described. In the subsequent sections the (meta)data types obtained from the transformation of resources into a merged ontology are also described.

We briefly describe:

- the resources that are integrated
- how the Integrated Fishery Ontology (IFO) is being built
- a mediation architecture to interface the fishery ontology service with the source information systems.

3 OUTLINE OF THE FOS PROJECT

3.1 Resources

The following resources have been singled out from the fishery information systems considered in the project:

the **oneFish** topic trees (about 1,800 topics), made up of *hierarchical topics* with brief summaries, identity codes and attached knowledge objects (documents, web sites, various metadata). The hierarchy (average depth: 3) is ordered by (at least) two different relations: *subtopic*, and *intersection between topics*, the last being notated with @, similarly to relations found in known subject directories like DMOZ. There is one 'backbone' tree consisting of five disjoint categories, called *worldviews* (*subjects, ecosystem, geography, species, administration*) and one worldview (*stakeholder*), maintained by the users of the community, containing own topics and topics that are also contained in the first four other categories (Figure 5). Alternative trees contain new 'conjunct' topics deriving from theintersection of topics belonging to different categories.

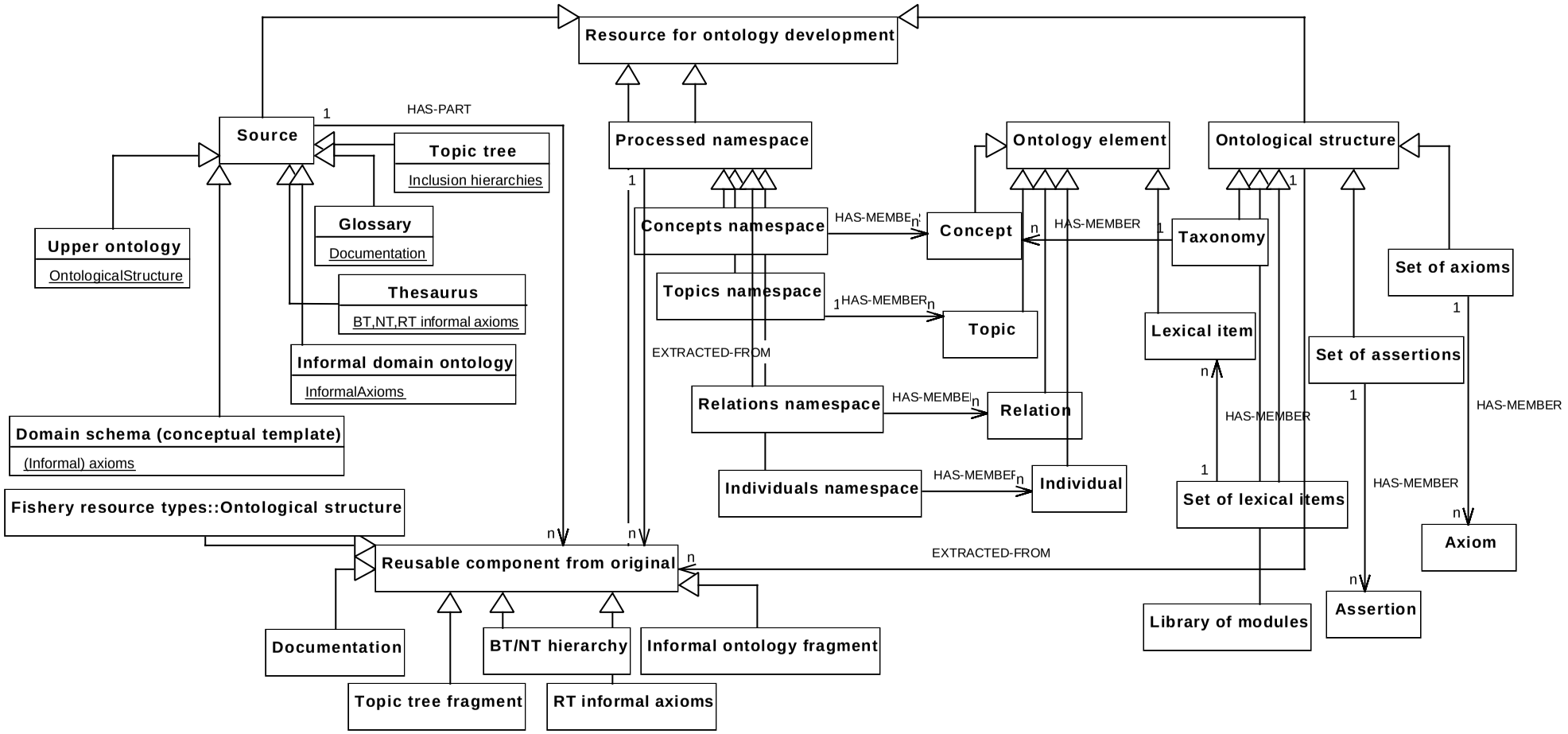

Fig. 1. A class diagram of the source data types taken into account

AGROVOC thesaurus (about 500 fishery-related descriptors), with thesaurus relations (*narrower term*, *related term*, *used for*) among descriptors, lexical relations among terms, terminological multilingual equivalents, and glosses (*scope notes*) for some of them.

ASFA thesaurus, similar to AGROVOC, but with about 10,000 descriptors.

FIGIS reference tables, with 100 to 200 top-level concepts, with a max depth of 4, and about 30,000 'objects' (mixed concepts and individuals), relations (specialised for each top category, but scarcely instantiated) and multilingual support. There are modules (*water areas*, *continental areas*, *biological entities*, *vessels*, *commodities*, *stocks*, etc.), also organised by 'views'.

In Figure 2 a diagram is sketched of the methodology used to extract and refine the informal data from the fishery information systems. The methodology is also described in the next sections.

3.2 Translation and refining of the components for IFO building

The (meta)data from the resources that have been singled out have been processed, in order to integrate them within a homogeneous environment, and with a clear assessment of their nature. In the following we list a set of guidelines that have been followed to translate and refine data components:

- A detailed evaluation of each source (find the schema -explicit or not- underlying the implementation of source data, then describe each data type both qualitatively and quantitatively) is performed.
- A language to represent the KB is chosen that hosts the integration activity. A description logic like DLR [9] is an ideal choice for its compatibility with the ontology integration framework.
- An ontology server is installed that supports DLR or compatible languages.
- Some data types from the sources (Figure 1) seem appropriate to be included in a preliminary prototype. The following steps are performed on them:
    - Discuss, refine and formalise FIGIS fishery conceptual schemas [12] to build a preliminary core ontology. Also the upper-level concepts from the source thesauri should be matched against the FIGIS conceptual schemas. This results in a *resource for core ontology development*.
    - Translate FIGIS reference tables: taxonomy, individuals, and local relations (to be transformed into formal axioms). This results in a *draft resource for domain ontology development*.
    - Reuse oneFish topic trees to design a preliminary architecture for IFO library. This architecture should match the preliminary core ontology. This results in a *resource for ontology library design*.

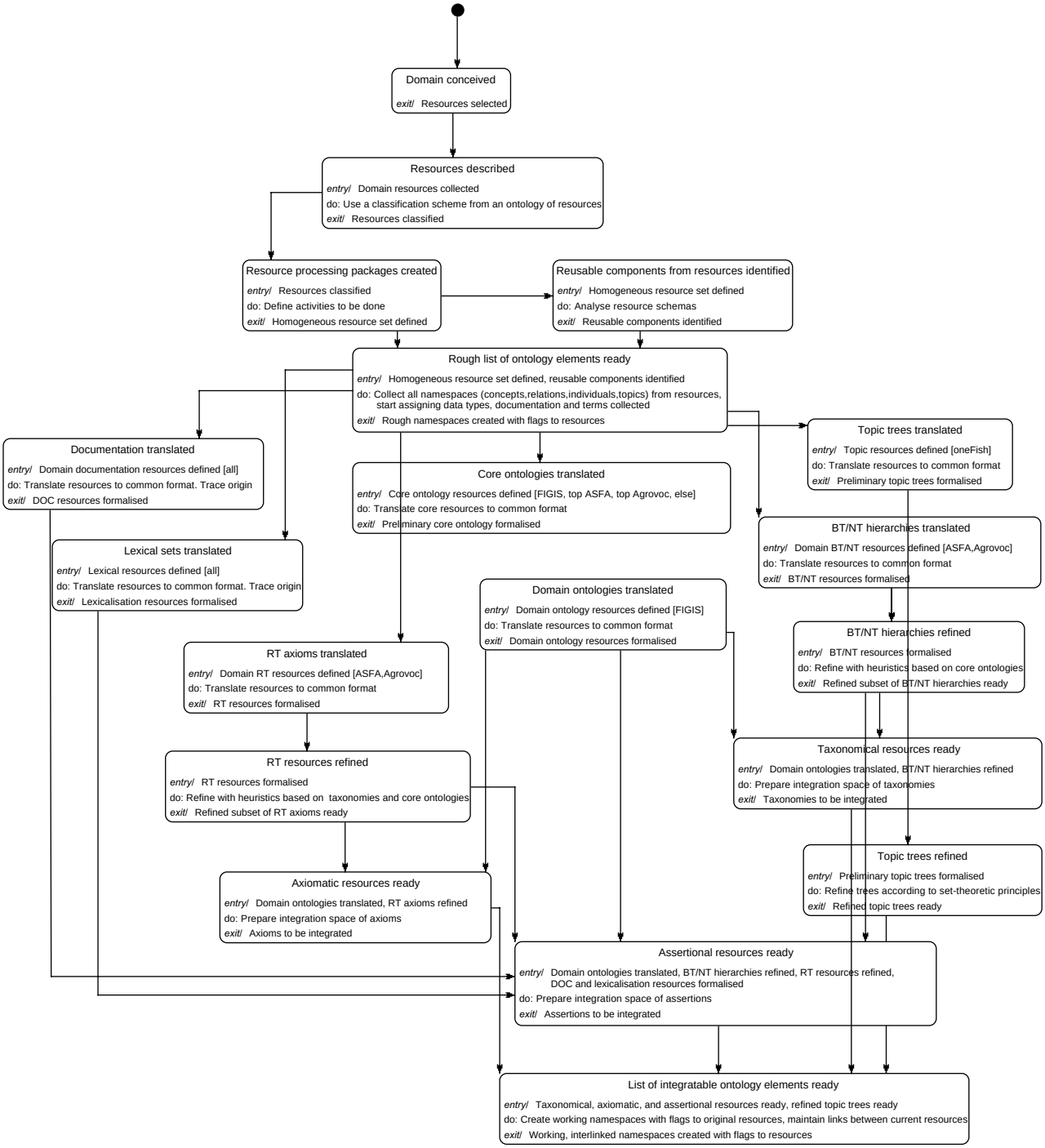

Fig. 2. A diagram of the methodology used to extract and refine the informal data

- Extract IS_A taxonomies from AGROVOC and ASFA BT/NT (*Broader Term/Narrower Term*) hierarchies. Heuristics from upper and core ontologies can be applied to clean up BT/NT hierarchies, for example, the following rule can be applied: *if a body part descriptor is NT of an organism descriptor, then this is probably* not *an IS_A use of NT* (probably it is a *part-of* relation). This results in *resources for core and domain taxonomies building*.
- Expand RT (*Related Term*) relations from AGROVOC and ASFA. Also non-IS_A BT/NT hierarchies could be refined (expanded) here. Heuristics can be applied here as well, for example, *if there exists a systematic relation between to concepts in the core ontology, and there exists a RT relations between two subconcepts of those concepts, then this is an indication for that relation to be the refinement of the RT one*. This results in *resources for core and domain axioms building*.
- Reuse UF (*Used For*) relations and (multi-)linguistic equivalents from all resources. Track must be kept of the context from which a linguistic item has been extracted. This results in *resources for ontology lexicalisation*.

3.3 Parallel tasks

In the following sections we outline the main steps to build the basic taxonomy, documentation, and architecture for the integrated fishery ontology.

3.3.1 Developing a fishery core ontology (FCO)

In this step, we pick up uppermost concepts and conceptual (categorisation) schemas from sources and integrate them with a certified top-level containing domain-independent concepts, relations and meta-properties. The resources needed for such a task are:

Upper ontology resources: the OntoClean upper level [8,18] (Figure 3) is a preferential choice for its compatibility with the methodology. For alternatives, see [13]. Moreover, various formal ontologies and standards for relations, and general lexical repositories like WordNet [14].

Core ontology resources: conceptual templates, (selected in the preliminary phases), relational database schemas, theoretical views on domain topics, domain standards, etc. An informal fishery core ontology (the FIGIS *composite concepts*) is shown in Figure 4.

In the context of core ontology development, some taxonomical branches (*core concepts*) have relevant conceptual integration issues that are being studied by ontological engineers and domain experts in close collaboration:

- *biological taxonomies*: difficult having a stable framework of reference (in principle, mapping from local taxonomies to a biological one is feasible, but in practice it could be not cost effective)
- *geographic regions*: use GIS as a stable framework of reference? geographic names?
- *institutions*: maybe automatic clustering of individuals through classification

- *fishing devices* (including vessels)
- *fishing and fish farming techniques* (plans and activity types)
- *farming systems* (sets of components)
- *fishery regulations* (norms)
- *fishery managament systems* (plans)
- *production centers*

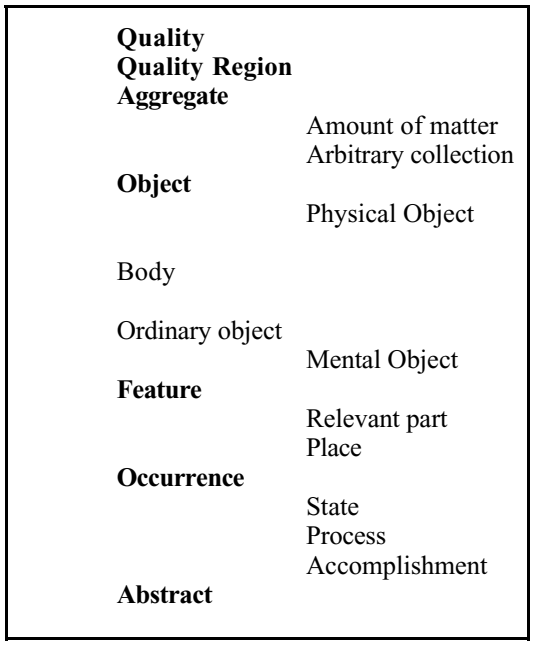

Fig. 3. The OntoClean top concepts

Development is performed as incremental loading and classification of upper and core level ontologies in the Ontology Server.

Another indirect resource that can be exploited to build the core ontology is the analysis of *systematic polysemies* (they have been already used in the mining of large medical thesauri, cf.[10]). A systematic polysemy is discovered when a relation exists between two senses of a term, and this relation is *relevant* for the domain that is being analysed. Consequently, if we find many polysemies with senses that have been conceptualised within the same concept pairs, this is an indication for a possible core ontology relation.

3.3.2 Building domain IS-A taxonomies

This phase deals with the integration of the resources for domain ontology development with the fishery core ontology (developed in the previous phase).

Resulting taxonomies could be either 'tolerated' or 'cleaned up'. Tolerance amounts to have widespread and unexplained polysemy for terms, but it is not time consuming. Cleaning is the most time consuming task, since a frequent scenario is the following: concept C from source S1 (C^S1) is in principle similar to a D^S2

(usually because they share one or more terms), but they actually occupy two taxonomical places that make them disjoint according to the upper or core ontology.

The ONIONS methodology [10] in this case suggests to axiomatise their glosses (cf. ⁄3.2.3, 3.3.3) and to check if their taxonomical position is correct. If it is not, then they are probably polysemous senses of the same term, and some alternative methods can be applied to relate those senses, to merge them, or to accept the conceptual split of the senses.

Some cleaning will be needed in any case to remove at least the major taxonomical clashes. This results into a *domain taxonomy*. Additional effort should be dedicated to distinguish:

Concepts vs individuals (heuristics applicable: country names, institutions, etc.).

Backbone concepts vs viewpoint concepts (roles, reified properties, contingent notions), cf. [7,8].

This eventually results into a refined domain taxonomy.

Fig. 4. The FIGIS *composite concepts*, used as a resource for core ontology development.

3.3.3 Collecting existing documentation and producing glosses

Available resources for ontology documentation are collected and associated as a kind of annotation (*gloss*) to domain concepts. Concepts lacking a gloss require a new one.

For core concepts and relations, besides existing glosses, an extensive description of their scope in the FCO is provided.

3.3.4 Designing a preliminary topic architecture

A preliminary topology for most general topics (to be used for ontology modularisation as well) is figured out. Here the following resources are reused: ontologies for topics (Welty s topic topology [15], topic maps standard [16], OnTopic principles [11]), semantic portals design [17], *oneFish* topic trees.

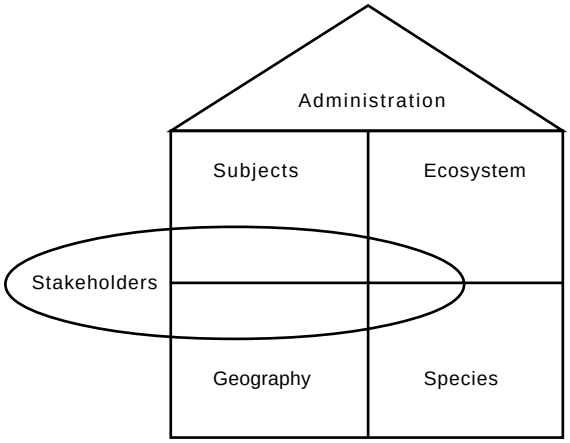

Fig. 5. Topic spaces (*"worldviews"*) in oneFish.

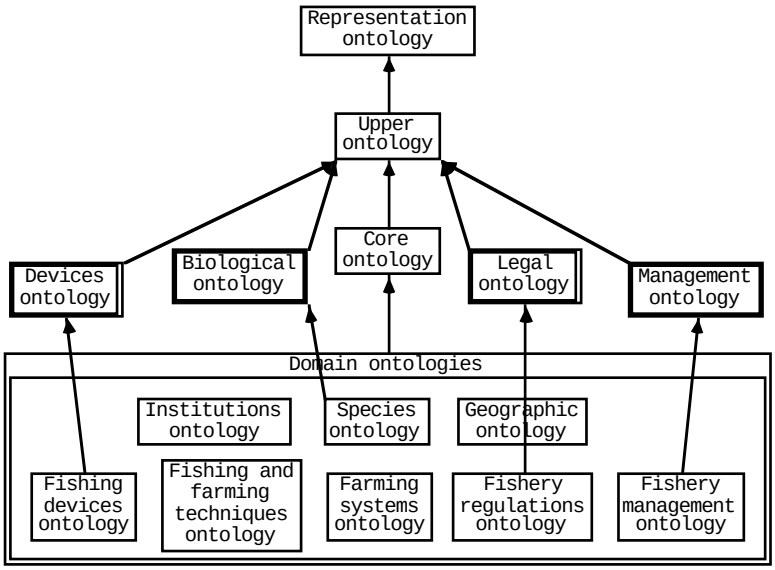

Fig. 6. An example architecture for the fishery ontology library. Double frames mean external ontologies.

The topic topology will be used both for maintaining the ontology library and for managing text indexing and retrieval. Figure 5 shows how the current topic spaces of oneFish are structured. Figure 6 shows an ontology-based architecture for the Integrated Fishery Ontology.

3.4 Building domain axioms

Once taxonomies are cleaned to a certain extent, documented, and divided into appropriate namespaces, activities aimed at raising the conceptual detail of the ontology can be started. The most important is the characterisation of domain concepts with axioms. In order to realise this, domain resources containing informal relationships, and (at least some) glosses from documentation are upgraded to the status of logical axioms.

Informal relationships can be found in thesauri (e.g. *related term*) as well as reference tables and topic trees. They are mined in order to understand:

1) if the axioms are applicable to all the subconcepts of the concept to which the axiom pertain, and
2) what quantification is applicable to those axioms: existential (necessary) or universal (contingent)?

This results into formal Domain Axioms. This axiom set is enhanced by axiomatising glosses. Here the ONIONS methodology [10] is applied to derive formal domain axioms from natural language descriptions. The typical technique consists in extracting terms, parsing them according to a dependency grammar, and applying core and upper ontologies to assign concepts and relations to the resulting dependency trees.

This activity is time-consuming, and semi-automatic techniques are still a research issue [13]. Scalability and approximate results are considered here.

The axioms obtained from informal relationships and glosses are revised according to the fishery core ontology developed so far.

3.5 Modularising ontology library according to topics

Following OnTopic methodology [11], dependency chains of core concepts are automatically generated and the existing preliminary topic topology is checked in order to produce a first version of the ontology library architecture. Dependency chains are also applied to derive indexing tags and boolean search spaces.

A dependency chain is the transitive closure of the logical depend-ons of a concept. The transitive closure is applied to the defining elements of a concept. Here a set of *relevance parameters* are applied in order to

3.6 Providing multi-lingual lexicalisation to elements in the ontology library

An integrated fishery ontology benefits from the existence of terms already related to concepts in the original resources, since they semi-automatically provide the so-called *lexicalisation* of concepts. On the other hand, having an integrated ontology also provides a powerful tool to check polysemous senses of terms, as well as to check consistency of UF thesaurus relations and consistency of multi-lingual equivalents.

3.7 A unified architecture

Figure 7 shows a simplified example architecture to support information brokering [6] or unified search after merging of fishery information systems by means of Fishery Ontology Service.

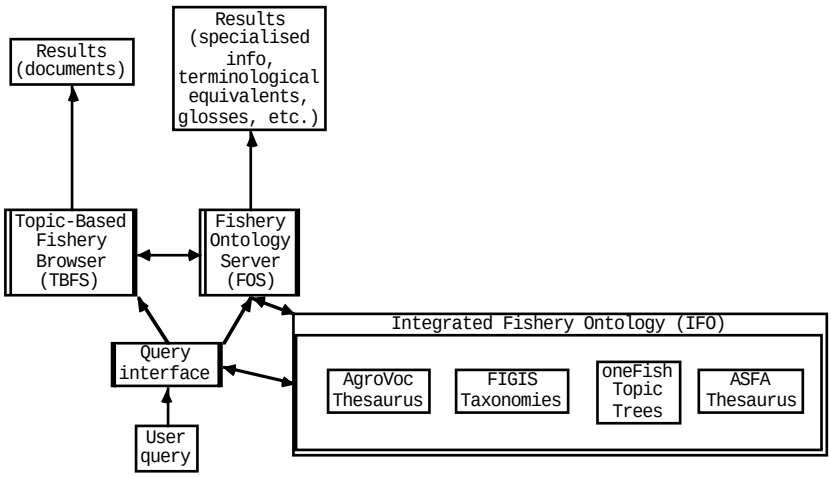

Fig. 7. A unified interface for interoperability after merging heterogeneous terminological resources in fishery.

The basic idea is that user queries, through a query interface, can be submitted to two kinds of servers: if the query aims at retrieving documents, a topic-based fishery agent rewrites the query in order to submit it to heterogeneous databases (*brokering*); if the query aims at finding specialised conceptual or terminological information, it is directed to the Fishery Ontology Server (FOS). In both cases, the query interface uses FOS. Query rewriting needs also mapping relations from the integrated fishery ontology to the source thesauri.

CONCLUSIONS

In this paper we have outlined some research solutions within the framework of ontology integration that are based on formal upper and core ontologies. Some details have been given on how informal schemata such as thesauri, reference tables, and topic trees can be reused and refined in order to be manipulated by ontology integration. Some hints have also been shown about the dependence of topic trees from ontologies, a promising research area for the semantic web.

In fact, the overall research issue underlying the FOS project is to provide a unified methodology of ontology integration and merging based on formal ontologies, ontology library design, topic trees building and maintainance, and efficient web search and indexing.

REFERENCES

[1] http://www.fao.org/fi
[2] http://www4.fao.org/asfa
[3] http://www.fao.org/agrovoc
[4] http://www.onefish.org
[5] http://www.ontoweb.org
[6] http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/theo-6/web-agent/www/i3.html
[7] Gangemi A, Guarino N, Masolo C, Oltramari A.: Understanding Top-Level Ontological Distinctions, in: H. Stuckenschmidt (ed), *Proceedings of the IJCAI 2001 Workshop on Ontologies and Information Sharing* (2001)
[8] Gangemi A, Guarino N, Oltramari A.: Conceptual Analysis of Lexical Taxonomies: The Case of WordNet Top-Level, in: C Welty, B Smith (eds.), *Proceedings of the 2001 Conference on Formal Ontology and Information Systems,* Amsterdam, IOS Press (2001)
[9] Calvanese D, De Giacomo G, Lenzerini M.: A Framework for Ontology Integration. Proceedings of 2001 Int. Semantic Web Working Symposium (SWWS 2001) (2001)
[10] Gangemi A, Pisanelli DM, Steve G.: An Overview of the ONIONS Project: Applying Ontologies to the Integration of Medical Terminologies. *Data and Knowledge Engineering*, 1999, vol.31, pp. 183-220 (1999)
[11] Gangemi A, Pisanelli DM, Steve G.: The OnTopic Methodology for Supporting Active Catalogues with Formal Ontologies. ISTC-CNR-OCMG Internal Report iii-01 (2001)
[12] Taconet M, Roux O: FIGIS, The Fisheries Global Information System.
[13] http://www.ontoweb.org/SIG
[14] Velardi P, Missikoff M, Fabriani P: Using Text Processing Techniques to Automatically Enrich a Domain Ontology, in: C Welty, B Smith (eds.), *Proceedings of the 2001 Conference on Formal Ontology and Information Systems,* Amsterdam, IOS Press (2001)
[15] Welty C, The Ontological Nature of Subject Taxonomies, N Guarino (ed.), *Proceedings of the First Conference on Formal Ontology and Information Systems,* Amsterdam, IOS Press (1998)
[16] Pepper S, The TAO of Topic Maps: http://www.gca.org/papers/xmleurope2000/papers/s11-01.html
[17] Stojanovic N, Maedche A, Staab S, Studer R, Sure Y: SEAL —A Framework for Developing SEmantic PortALs
[18] Oltramari A., Gangemi A, Guarino N, Masolo C,: Restructuring WordNet's Top-Level: The *OntoClean* approach, in K Simov (ed): *Proceedings of the The LREC2002 Workshop on Ontologies and Text*, Las Palmas (2002)

Finding and specifying relations between ontology versions

Michel Klein[1] and **Atanas Kiryakov**[2] and **Damyan Ognyanoff**[3] and **Dieter Fensel**[4]

Abstract. Interoperability between different existing ontologies is import to leverage the use of ontologies. However, the interoperability between different *versions* of ontologies is at least as important. Especially when ontologies are used in a distributed and dynamic context as the Web, we can expect that ontologies will rapidly evolve and thus may cause incompatibilities. This paper describes a system that helps to keep different versions of web-based ontologies interoperable. To achieve this, the system allows ontology engineers to compare versions of ontology and to specify the conceptual relations between the different versions of concepts. Internally, the system maintains the transformations between ontologies, some meta-data about the version, as well as the conceptual relation between concepts in different versions. This paper briefly describes the system, presents the mechanism that we used to find and classify changes in RDF-based ontologies, and discusses how this may be used to specify relations between ontologies that improve their interoperability.

1 ONTOLOGY EVOLUTION THREATENS INTEROPERABILITY

Ontologies have become popular because of their promise of knowledge sharing and reuse [10]. Interoperability between ontologies is an important issue, because the reuse of knowledge often implies that different existing ontologies are used together. This requires that the knowledge represented in the ontologies is not conflicting. However, ontology interoperability is not only important between different existing ontologies, it is also an issue between different *versions* of an ontology. This is especially relevant when ontologies are used in the context of the Semantic Web [5].

In this vision, ontologies have a role in defining and relating concepts that are used to describe data on the web. The distributed and dynamic character of the web will cause that many versions and variants of ontologies will arise. Ontologies are often developed by several persons and continue to evolve over time. Moreover, domain changes, adaptations to different tasks, or changes in the conceptualization might cause modifications of the ontology. This will likely cause incompatibilities in the applications and ontologies that refer to them, and will give wrong interpretations to data or make data inaccessible [11].

To handle ontology changes, a change management system is needed that keeps track of changes and versions of ontologies. Moreover, it is necessary to maintain the links between the versions and variants that specify the relations and updates between the versions. These links can be used to re-interpret data and knowledge under different versions. The ontologies and their relations together form a *web* of ontologies. The specification of these links is thus very important.

In this paper, we present a web-based system that supports the user in specifying the conceptual relation between version of concepts. The system, called OntoView, also maintains those links, together with the transformations between them. It use them to provide a transparent interface to different versions of ontologies, both at a specification level as at a conceptual level. It can also export the differences between versions as separate "mapping ontologies", which can be used as adapters for the re-interpretation of data and other ontologies. The goal of this system is not to provide a central registry for ontologies, but to allow ontology engineers to store their versions and variants of ontologies and relate them to other (possibly remote) ontologies. The resulting mapping relations between versions can also be exported and used outside the system.

Most of the ideas underlying the versioning system are not depending on a specific ontology language. However, the implementation of specific parts of the system will be dependent on the used ontology language, for example the mechanism to detect changes. Throughout this article, we will use DAML+OIL[5] [8, 9] and RDF Schema (RDFS) [7] as ontology languages. These two languages are widely considered as basis for future ontology languages for the Web.

The rest of the paper is organized as follows. In the next section, we discuss some issues about update relations between ontologies. In section 3, we give an overview of the versioning system and describe its the main functions. Section 4 describes the main feature of the system: comparing ontologies. In that section, we explain the mechanism we used to find changes in RDF-based ontologies and present some of the rules that we used to encode change types. Finally, we conclude the paper in section 6.

2 THE UPDATE RELATION BETWEEN ONTOLOGIES

There are three important aspects to discuss when considering an update relation between ontologies. First, this is **the difference between update relations and conceptual relations inside an ontology**.

Ontologies usually consist of a set of class (or concept) definitions, property definitions and axioms about them. The classes, properties and axioms are related to each other and together form a model of a part of the world. A change constitutes a new version of the ontology. This new version defines an orthogonal relation between the

[1] Vrije Universiteit Amsterdam, michel.klein@cs.vu.nl
[2] OntoText, Sofia, Atanas.Kiryakov@sirma.bg
[3] OntoText, Sofia, damyan@sirma.bg
[4] Vrije Universiteit Amsterdam, dieter@cs.vu.nl

[5] Available from `http://www.daml.org/language/`

definitions of concepts and properties in the original version of the ontology and those in the new version. This is depicted in Figure 1.

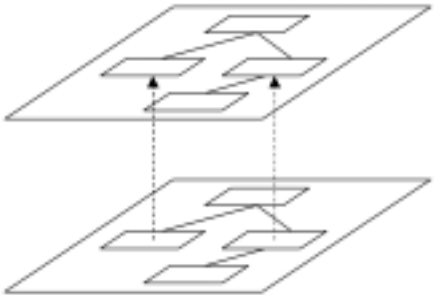

Figure 1. Orthogonal relations between classes in two version of an ontology (dashed arrows)

The relations between concepts inside an ontology, e.g. between class A and class B, is thus a fundamentally different relation from the update relation between two versions of a concept, e.g. between class $A_{1.0}$ and class $A_{2.0}$. In the first case, the relation is a purely conceptual relation in the domain; in the second case, however, the relation describes meta-information about the change of the concept.

Nevertheless, two version of a concept still have *some* conceptual relation. This relation, however, is not determined by the update itself, but accompanying information of an update relation. There are other characteristics of an update relation, too. We distinguish the following properties that can be associated with an update relation:

- **transformation** or **actual change**: a specification of what has actually changed in an ontological definition, specified by a set of change operations (cf. [1]), e.g., change of a restriction on a property, addition of a class, removal of a property, etc.;
- **conceptual relation**: the logical relation between constructs in the two versions of the ontology, e.g., specified by equivalence relations, subsumption relations, or logical rules;
- descriptive meta-data like **date**, **author**, and **intention** of the update: this describes the when, who and why of the change;
- **valid context**: a description of the context in which the update is valid. In its simplest form, this might consist of the date when the change is valid in the real world, conform to *valid date* in temporal databases [15] (in this terminology, the "date" in the descriptive meta-data is called *transaction date*). More extensive descriptions of the context, in various degrees of formality, are also possible.

A well-designed ontology change specification mechanism should take all these characteristics into account.

Another issue to discuss about ontology updates is the **possible discrepancy between changes in the specification and changes the conceptualization**. We have seen that a ontology is a *specification* of a *conceptualization*. The actual specification of concepts and properties is thus a *specific representation* of the conceptualization: the same concepts could also have been specified differently. Hence, a change in the specification does not necessarily coincide with a change in the conceptualization [11], and changes in the specification of an ontology are not per definition ontological changes.

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

The intention of a change is made explicit by categorizing them into the following categories [16]:

- **conceptual change**: a change in the way a domain is interpreted (conceptualized), which results in different ontological concepts or different relations between those concepts;
- **explication change**: a change in the way the conceptualization is specified, without changing the conceptualization itself.

A change cannot be automatically classified as belonging to one of these categories, because it is basically a decision of the modeler. However, heuristics can be applied to suggest the effects of changes. We will discuss that later on.

A third, somewhat different, aspect of an update is the **packaging of changes**, i.e., the way in which updates are applied to an ontology. This is an important practical issue for the development of an ontology change management system.

We can distinguish two different dimensions with respect to the packaging of the change specification. One dimension is the *granularity* of the specification: this can be either the level of a single "definition" or the level of a "file" as a whole.

The second dimension is the *method* of specification. There are several methods thinkable:

- a "transformation specification": an update specified by a list of change operations (e.g., add A, change B, delete C);
- a "replacement": an update specified by replacing the old version of a concept or an ontology with a new version; this is an implicit change specification;
- a "mapping": an update specified as a mapping between the original ontology and another one. Although this is not a update in the regular sense, an explicit mapping to another ontology can be considered as an update to the viewpoint of that ontology.

This gives several possible change specifications. For example, a change can be specified individually, as a mapping between one specific definition in one ontology and another definition in another ontology, but it can also be done at a file level, by defining the transformation of the ontology.

Notice that the packaging methods are not equivalent, i.e., they do not give the same information about the update relation. It is clear that the mapping provides a conceptual relation between versions of concepts that is not specified in a transformation.

3 GENERAL DESCRIPTION OF ONTOVIEW

OntoView is a web-based system under development that provides support for the versioning of online ontologies, which might help to solve some of the problems of evolving ontologies on the web. Its main function is to help the a user to manage changes in ontologies and keep ontology versions as much interoperable as possible. It does that by comparing versions of ontologies and highlighting the differences. It then allows the users to specify the conceptual relation between the different versions of concepts. This function is described more extensively in the next section.

It also provides a transparent interface to arbitrary versions of ontologies. To achieve this, the system maintains an internal specification of the relation between the different variants of ontologies, with the aspects that were defined in section 2: it keeps track of the **meta-data**, the **conceptual relations** between constructs in the ontologies and the **transformations** between them.

OntoView is inspired by the Concurrent Versioning System CVS [4], which is used in software development to allow collaborative development of source code. The first implementation is also based on CVS and its web-interface CVSWeb[6]. However, during the ongoing development of the system, we are gradually shifting to a complete new implementation that will be build on a solid storage system for ontologies, e.g., Sesame[7].

Besides the ontology comparison feature, the system has the following functions:

- **Reading changes and ontologies.** OntoView will accept changes and ontologies via several methods. Currently, ontologies can be read in as a whole, either by providing a URL or by uploading them to the system. The user has to specify whether the provided ontology is new or that it should be considered as an update to an already known ontology. In the first case, the user also has to provide a "location" for the ontology in the hierarchical structure of the OntoView system.
  Then, the user is guided through a short process in which he is asked to supply the meta-data of the version (as far as this can not be derived automatically, such as the date and user), to characterize the types of the changes (see below in section 4), and to decide about the identifier of the ontology.
  In the future, OntoView will also accept changes by reading in transformations, mapping ontologies, and updates to individual definitions. These update methods provides the system with different information than the method described above. For that reason, this also requires an adaptation of the process in which the user gives additional information.
- **Identification.** Identification of versions of ontologies is very important. Ontologies describe a consensual view on a part of the world and function as reference for that specific conceptualization. Therefore, they should have a unique and stable identification. A human, agent or system that conforms to a specific ontology, should be able to refer to it unambiguously.
  Usually, the XML Namespace mechanism [6] is used for the identification of web-based ontologies. This means that an ontology is identified by a URI, i.e. a unique pointer on the web. In practice, people tend to use the location (the URL) of the ontology file on the web as identifier. OntoView also uses the namespace mechanism for identification, but does not necessarily use the location of the ontology file. If a change does not constitute a conceptual change, the new version gets a new location, but does not get a new identifier. For example, the location of an ontology can change from "../example/1.0/rev0" to "../example/1.0/rev1", while the identifier is still "../example/1.0".
  OntoView supports two ways of persistent and unique identification of web-based ontologies. First, it can in itself guarantee the uniqueness and persistency of namespaces that start with "http://ontoview.org/", because the system is located at the domain `ontoview.org`. Second, because the location and identification of ontologies are only loosely coupled, it can also store ontologies with arbitrary namespaces. In this case, the ontology engineer is responsible for guaranteeing the uniqueness. The ontologies with arbitrary namespaces are not directly retrievable by their namespace, but can be accessed via a search function.
- **Analyzing effects of changes.** Changes in ontologies do not only affect the data and applications that use them, but they can also have unintended, unexpected and unforeseeable consequences in the ontology itself [13].
  OntoView provides some basic support for the analysis of these effects. First, on request it can also highlight the places in the ontology where conceptually changed concepts or properties are used. For example, if a property "hasChild" is changed, it will highlight the definition of the class "Mother", which uses the property "hasChild". In the future, this function should also exploit the transitivity of properties to show the propagation of possible changes through the ontology.
  Further, we expect to extend the system with a reasoner to automatically verify the changes and the specified conceptual relations between versions. For example, we could couple the system with FaCT [3] and exploit the Description Logic semantics of DAML+OIL to check the consistency of the ontology and look for unexpected implied relations.
- **Exporting changes.** The main advantage of storing the conceptual relations between versions of concepts and properties is the ability to use these relations for the re-interpretation of data and other ontologies that use the changed ontology. To facilitate this, OntoView can export differences between ontologies as separate mapping ontologies, which can be used as adapters for data sources or other ontologies. They only provide a partial mapping, because not all changes can be specified conceptually.
  The exported mapping ontologies are represented with the standard constructs of the ontology langauge. Because in OntoView the conceptual relation and the actual transformation are stored separately, it is not necessary to extend the ontology language with more advanced mapping- or transformation primitives than those already available.
  The meta-data about the ontology update is specified as a set of properties of the conceptual relations themselves. In DAML+OIL, this meant that we had to re-ify the mapping statements.[8] This method has two advantages. First, when specified over re-ified statements, the meta-data does not interfere with the actual ontological knowledge, as would be the case when meta-data is specified as characteristics of classes and properties. Second, because the meta-data is data about the *mappings themselves*, agents or systems that understand the meta-data can use this to decide which mappings are applicable in a specific context and which are not.
  In the future, it should also be possible to export *transformations* between two versions of an ontology. A transformation is a complete specification of all the change operations. This can be used to re-execute changes and to update ontologies that have some overlap with the versioned ontology in exactly the same way as the original one. However, transformations facilitates data re-interpretations only to a very small extent. A mapping ontology provides better re-interpretation, because it also captures human knowledge about the relations.

4 COMPARING ONTOLOGIES

One of the central features of OntoView is the ability to compare ontologies at a structural level. The comparison function is inspired by UNIX `diff`, but the implementation is quite different. Standard `diff` compares file version at line-level, highlighting the lines that

[6] Available from `http://stud.fh-heilbronn.de/~zeller/cgi/cvsweb.cgi/`

[7] A demo is available at `http://sesame.aidministrator.nl`

[8] The DAML+OIL semantics do not currently cover reification because of the undecidability of second-order logic. However, there is an awareness that use reification for "tagging" purposes — as we do — is different from full second-order logic. See `http://www.daml.org/language/features.html`.

textually differ in two versions. OntoView, in contrast, compares version of ontologies at a *structural* level, showing which definitions of ontological concepts or properties are changed. An example of such a graphical comparison of two versions of a DAML+OIL ontology is depicted in Figure 2.[9]

4.1 Types of change

The comparison function distinguishes between the following types of change:

- Non-logical change, e.g. in a natural language description. In DAML+OIL, this are changes in the rdfs:label of an concept or property, or in a comment inside a definition. An example is the first highlighted change in Figure 2 (class "Animal').
- Logical definition change. This is a change in the definition of a concept that affects its formal semantics. Examples of such changes are alterations of subClassOf, domain, or range statements. Additions or deletions of local property restrictions in a class are also logical changes. The second and third change in the figure is (class "Male" and property "hasParent") are examples of such changes.
- Identifier change. This is the case when a concept or property is given a new identifier, i.e. a renaming.
- Addition of definitions.
- Deletion of definitions.

Most of these changes can be detected completely automatically, except for the identifier change. Each type of change is highlighted in a different color, and the actually changed lines are printed in boldface. We describe the mechanism that we use to detect and classify changes in the next paragraphs.

4.2 Detecting changes

There are two main problems with the detection of changes in ontologies. The first problem is the abstraction level at which changes should be detected. Abstraction is necessary to distinguish between changes in the representation that affect the meaning, and those that don't influence the meaning. It is often possible to represent the same ontological definition in different ways. For example, in RDF Schema, there are several ways to define a class:

```
<rdfs:Class rdf:ID="ExampleClass"/>
```

or:

```
<rdf:Description rdf:ID="ExampleClass">
  <rdf:type rdf:resource="...chema#Class"/>
</rdf:Description>
```

Both are valid ways to define a class and have exactly the same meaning. Such a change in the representation would not change the ontology. Thus, detecting changes in the *representation* alone is not sufficient.

However abstracting too far can also be a problem: considering the *logical meaning* only is not enough. In [2] is shown that different sets of ontological definitions can yield the same set of logical axioms. Although the logical meaning is not changed in such cases, the ontology definitely is. Finding the right level of abstraction is thus important.

Second, even when we found the correct level of abstraction for change detection, the conceptual implication of such a change is not yet clear. Because of the difference between conceptual changes and explication changes (as described in section 2), it is not possible to derive the conceptual consequence of a change completely on basis of the visible change only (i.e., the changes in the definitions of concepts and properties). Heuristics can be used to suggest conceptual consequences, but the intention of the engineer determines the actual conceptual relation between versions of concepts.

In the next two sections, we explain the algorithm that we used to compare ontologies at the correct abstraction level, and how users can specify the conceptual implication of changes.

4.3 Rules for changes

The algorithm uses the fact that the RDF data model [12] underlies a number of popular ontology languages, including RDF Schema and DAML+OIL. The RDF data model basically consists of triples of the form `<subject, predicate, object>`, which can be linked by using the object of one triple as the subject of another. There are several syntaxes available for RDF statement, but they all boil down to the same data model. An set of related RDF statements can be represented as a graph with nodes and edges. For example, consider the following DAML+OIL definition of a class "Person".

```
<daml:Class rdf:ID="Person">
  <rdfs:subClassOf rdf:resource="#Animal"/>
  <rdfs:subClassOf>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#hasParent"/>
      <daml:toClass rdf:resource="#Person"/>
    </daml:Restriction>
  </rdfs:subClassOf>
</daml:Class>
```

When interpreted as a DAML+OIL definition, it states that a "Person" is a kind of "Animal" and that the instances of its hasParent relation should be of type "Person". However, for our algorithm, we are first of all interested in the RDF interpretation of it. That is, we only look at the triples that are specified, ignoring the DAML+OIL meaning of the statements. Interpreted as RDF, the above definition results in the following set of triples:

subject	predicate	object
Person	rdf:type	daml:Class
Person	rdfs:subClassOf	Animal
Person	rdfs:subClassOf	*anon-resource*
anon-resource	rdf:type	daml:Restriction
anon-resource	daml:onProperty	hasParent
anon-resource	daml:toClass	Person

This triple set is depicted as a graph in Figure 3. In this figure, the nodes are resources that function as subject or object of statements, whereas the arrows represent properties.

The algorithm that we developed to detect changes is the following. We first split the document at the first level of the XML document. This groups the statements by their intended "definition". The definitions are then parsed into RDF triples, which results in a set of small graphs. Each of these graphs represent a specific definition of a concept or a property, and each graph can be identified with the identifier of the concept or the property that it represents.

[9] This example is based on fictive changes to the DAML example ontology, available from `http://www.daml.org/2001/03/daml+oil-ex.daml`.

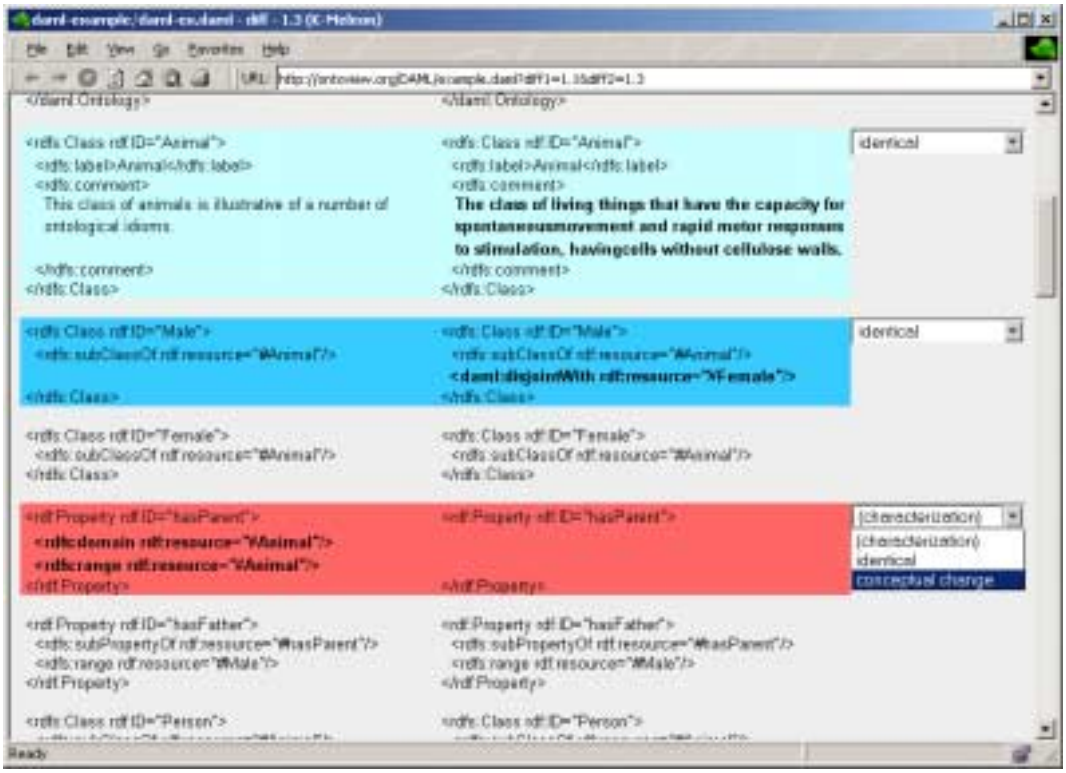

Figure 2. Comparing two ontologies

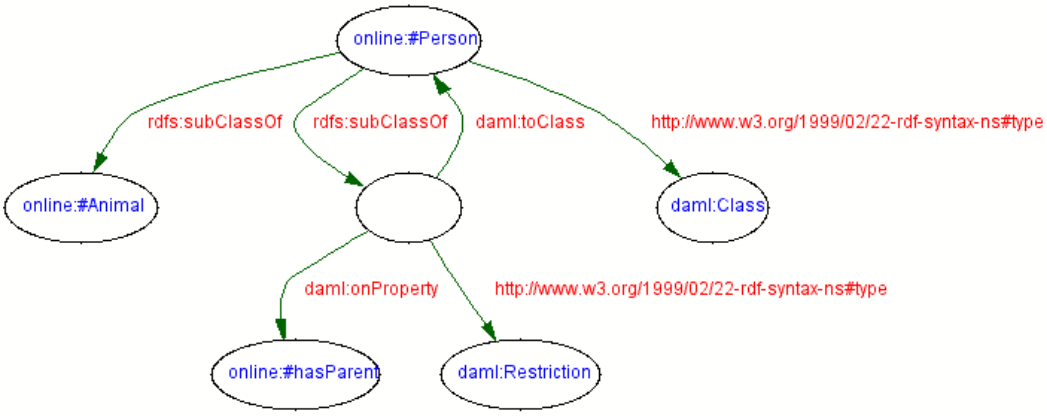

Figure 3. An RDF graph of a DAML class definition.

Then, we locate for each graph in the new version the corresponding graph in the previous version of the ontology. Those sets of graphs are then checked according to a number of rules. Those rules specify the "required" changes in the triples set (i.e., the graph) for a specific type of change, as described in section 4.1.

Rules have the following format:

```
IF exist:old
     <A, Y, Z>*
   not-exist:new
     <X, Y, Z>
THEN change-type A
```

They specify a set of triples that should exists in one specific version, and a set that should not exists in another version (or the other way around) to signal a specific type of change. With this rule mechanism, we were able to specify almost all types of change, except the identifier change. Here we also used some heuristics, based on the location of the definition in the file. We list two example rules below.

A change in the value of a local property:

```
IF exist:old
     <X, rdfs:subClassOf, Y1>
     <Y1, rdf:type, daml:#Restriction>
     <Y1, daml:onProperty, Y2>
     <Y1, daml:toClass, Z>
   not-exist:new
     <Y1, daml:toClass, Z>
THEN logicalChange.localPropertyValue X
```

A change in the property type:

```
IF exist:old
     <X, rdf:type, rdf:#Property>
     <X, rdf:type, daml:#UniqueProperty>
   not-exist:new
     <X, rdf:type, daml:#UniqueProperty>
THEN logicalChange.propertytype X
```

The rules are specific for a particular RDF-based ontology language (in this case DAML+OIL), because they encode the interpretation of the semantics of the language for which they are intended. For another language other rules would have been necessary to specify other differences in interpretation. The semantics of the language are thus encoded in the rules. For example, the last example not looks at changes in values of predicates (as the first does), but at a change in the type of property. This is a change that is related to the specific semantics of DAML+OIL.

Also, notice that the mechanism relies on the "materialization" of all `rdf:type` statements that are encoded in the ontology (sometimes called "knowledge compilation"). The last example depend on the existence of a statement `<X,rdf:type,rdf:#Property>`. However, this statement can only be derived using the semantics of the `rdfs:subPropertyOf` statement, which — informally spoken[10] — says that if a property is an instance of type X, then it is also an instance of the supertypes of X. The application of the rules thus has to be preceded by the materialization of the superclass- and superproperty hierarchies in the ontology. For this materialization, the entailment rules in the RDF Model Theory[11] can be used.

4.4 Specifying the conceptual implication of changes

The comparison function also allows the user to *characterize* the conceptual implication of the changes. For the first three types of changes that were listed in section 4.1, the user is given the option to label them either as "identical" (i.e., the change is an explication change), or as "conceptual change", using the drop-down list next to the definition (Figure 2). In the latter case, the user can specify the conceptual relation between the two version of the concept. For example, the change in the definition of "hasParent" could by characterized with the relation $\texttt{hasParent}_{1.1}$ `subPropertyOf` $\texttt{hasParent}_{1.3}$.

5 DISCUSSION

There are a few other issues and choices about the design of the system that we want to discuss. First, we purposely do not provide support for finding mappings between arbitrary ontologies. The intention of our system is to provide users with a system to manage versions of ontologies and maintain their relations. Finding the relations is a different task. However, it might be possible to incorporate this function in a future version of the system, e.g. by interfacing it with a ontology mapping tool.

Another issue is the visualization of the changes. The current versions shows the changes by highlighting the textual definitions that are changed. More advanced visualization techniques are possible. For example, one could think of techniques that render ontologies in a graphical representation and highlight the changes in the picture.

We did not yet specify the way in which a "valid context" is described. Such a context will have several dimensions, of which "time" is only one. This is something what still has to be done. Without such a specification, it is difficult to assess the validness of a conceptual relation between concepts in different versions. We can assume that such a relation is at least valid between two successive versions, but we do not know whether such mapping is allowed to "propagate" via other mappings to other ontologies. Research on this is necessary.

A situation in which versioning support is also necessary is the collaborative development of an ontology [14]. We think that OntoView is also useful in this situation, especially because all the conceptual implications of versions have to be characterized individually by users. This integrates the conflict resolution in the update procedure.

A side remark about the use of a versioning system for collaborative ontology development is that this gives an evolutionary way of ontology building. Each person can have its own conceptualization, which is conceptually linked to the conceptualizations of others. In this sense, the combination of versions and adaptations in itself forms a *shared* conceptualization of a domain.

Finally, we want to mention that the system is still under construction. In section 3 we extensively depicted the foreseen functionality of OntoView. However, as became clear of some of the descriptions, not everything is already realized. The basis functions are implemented, but a number of more advanced functions are still being developed.

6 CONCLUSION

When ontologies are used in a distributed and dynamic context, versioning support is essential ingredient to maintain interoperability. In this paper we have analyzed the versioning relation, described its aspects, and depicted a system that provides support for the versioning of online ontologies.

We described how this systems supports helps users to compare ontologies, and what the problems and challenges are. We presented a algorithm to perform a comparison for RDF-based ontologies. This algorithm doesn't operate on the representation of the ontology, but on the data model that is underlying the representation. By grouping the RDF-triples per definition, we still retained the necessary representational knowledge. We also explained how users can specify the conceptual implication of changes to help interoperability. This honors the fact that it is not possible to derive all conceptual implications of changes automatically.

The analysis of a versioning relation between ontologies revealed several dimensions of it. In the system that we described, all these dimensions are maintained separately: the descriptive **meta-data**, the **conceptual relations** between constructs in the ontologies, and the **transformations** between the ontologies themselves. This multi-dimensional specification allows both complete transformations of ontology representations and partial data re-interpretations, which help interoperability. The conceptual differences can be exported and used stand alone, for example to adapt data sources and ontologies.

The described system is not yet finished and should be developed further. We believe that it will significantly simplify the change management of ontologies and thus help the interoperability of evolving ontologies on the web.

[10] The precise semantics of RDF Schema are still under discussion.
[11] `http://www.w3.org/TR/rdf-mt/`

REFERENCES

[1] Jay Banerjee, Won Kim, Hyoung-Joo Kim, and Henry F. Korth, 'Semantics and Implementation of Schema Evolution in Object-Oriented Databases', *SIGMOD Record (Proc. Conf. on Management of Data)*, **16**(3), 311–322, (May 1987).

[2] Sean Bechhofer, Carole Goble, and Ian Horrocks, 'DAML+OIL is not enough', in *Proceedings of the International Semantic Web Working Symposium (SWWS)*, Stanford University, California, USA, (July 30 – August 1, 2001).

[3] S. Bechhofer, I. Horrocks, P. F. Patel-Schneider, and S. Tessaris, 'A proposal for a description logic interface', in *Proceedings of the International Workshop on Description Logics (DL'99)*, eds., P. Lambrix, A. Borgida, M. Lenzerini, R. Möller, and P. Patel-Schneider, pp. 33–36, Linköping, Sweden, (July 30 – August 1 1999).

[4] Brian Berliner, 'CVS II: Parallelizing software development', in *Proceedings of the Winter 1990 USENIX Conference*, ed., USENIX Association, pp. 341–352, Washington, DC, USA, (January 22–26, 1990). USENIX.

[5] Tim Berners-Lee, Jim Hendler, and Ora Lassila, 'The semantic web', *Scientific American*, **284**(5), 34–43, (May 2001).

[6] Tim Bray, Dave Hollander, and Andrew Layman. Namespaces in XML. `http://www.w3.org/TR/REC-xml-names/`, January 1999.

[7] D. Brickley and R. V. Guha, 'Resource Description Framework (RDF) Schema Specification 1.0', Candidate recommendation, World Wide Web Consortium, (March 2000).

[8] Dieter Fensel, Ian Horrocks, Frank van Harmelen, Stefan Decker, Michael Erdmann, and Michel Klein, 'OIL in a nutshell', in *Knowledge Engineering and Knowledge Management; Methods, Models and Tools, Proceedings of the 12th International Conference EKAW 2000*, eds., Rose Dieng and Olivier Corby, number LNCS 1937 in Lecture Notes in Artificial Intelligence, pp. 1–16, Juan-les-Pins, France, (October 2–6, 2000). Springer-Verlag.

[9] Dieter Fensel and Mark A. Musen, 'The semantic web: A new brain for humanity', *IEEE Intelligent Systems*, **16**(2), (2001).

[10] T. R. Gruber, 'A translation approach to portable ontology specifications', *Knowledge Acquisition*, **5**(2), (1993).

[11] Michel Klein and Dieter Fensel, 'Ontology versioning for the Semantic Web', in *Proceedings of the International Semantic Web Working Symposium (SWWS)*, Stanford University, California, USA, (July 30 – August 1, 2001).

[12] O. Lassila and R. R. Swick, 'Resource Description Framework (RDF): Model and Syntax Specification', Recommendation, World Wide Web Consortium, (February 1999). See http://www.w3.org/TR/REC-rdf-syntax/.

[13] Deborah L. McGuinness, Richard Fikes, James Rice, and Steve Wilder, 'An environment for merging and testing large ontologies', in *KR2000: Principles of Knowledge Representation and Reasoning*, eds., Anthony G. Cohn, Fausto Giunchiglia, and Bart Selman, pp. 483–493, San Francisco, (2000). Morgan Kaufmann.

[14] Helena Sofia Pinto and Jo ao Pavão Martins, 'Evolving ontologies in distributed and dynamic settings', in *Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning (KR2002)*, Toulouse, France, (April 22–25, 2002).

[15] John F. Roddick, 'A survey of schema versioning issues for database systems', *Information and Software Technology*, **37**(7), 383–393, (1995).

[16] Pepijn R. S. Visser, Dean M. Jones, T. J. M. Bench-Capon, and M. J. R. Shave, 'An analysis of ontological mismatches: Heterogeneity versus interoperability', in *AAAI 1997 Spring Symposium on Ontological Engineering*, Stanford, USA, (1997).

Ontology Theory

Christopher Menzel[1]

Abstract.
Ontology today is in many ways in a state similar to that of analysis in the late 18th century prior to arithmetization: it lacks the sort rigorous theoretical foundations needed to elevate ontology to the level of a genuine scientific discipline. This paper attempts to make some first steps toward te development of such foundations. Specifically, starting with some basic intuitions about ontologies and their content, I develop an expressively rich framework capable of treating ontologies as theoretical objects whose properties and logical interconnections — notably, potential for integration — we can clearly define and study.

1 Introduction

Ontology today is in a state similar to that of analysis in the late 18th century. The practical power of the calculus had been convincingly demonstrated in the work or Newton and his great successors. Moreover, the field of real analysis itself had seen an explosion of creativity, exemplified most notably in the work of Euler. However, Euler's own work also revealed worrisome foundational problems. For techniques used with great success in one instance to prove deep and dramatic theorems in another instance could lead to absurdities, e.g., that the value of certain monotonically increasing infinite series was -1. Such results led to a conceptual crisis — how can any results be trusted when the methods that generate them can lead to error?

This crisis was addressed, and successfully eliminated, by the development of rigorous foundation for analysis — widely known as the arithmetization of analysis — by Cauchy, Weierstrass, Bolzano, and others in the early 19th century. Building on the sound foundation of number theory, mathematicians replaced the intuitive but undefined notions of analysis — limit, continuity, series, integration, real number etc. — with clearly defined counterparts (e.g., the now-familiar ϵ, δ definition of limit) and banished unruly notions like that of an infinitesimal altogether.[2] With these solid underpinnings in place, mathematicians were able to identify clear conditions of applicability for their analytic methods that prevented the derivation of absurdities without limiting their ability to prove desirable results.

A similar foundation is needed in the study of ontologies. As with analysis prior to arithmetization, the potential of ontologies is evident, but the fundamental notions remain largely intuitive; notably, there is no precise characterization of the notion of an ontology, nor what it is for two ontologies to be intergrated. What we need, then, is our own "arithmetization" — in a nutshell, we need *ontology theory*: a mathematical framework, akin to number theory or modern analysis, that enables us to characterize the notion of an ontology formally and develop accounts of their properties and the various ways in which one ontology can be related to another. Note also that the framework itself might not be used as it stands for any actual ontology integration work. It is in the respect analogous to computability theory. No one actually programs turing machines (except as a heuristic exercise). Rather, the notion provides a model of computation that serves as a foundation for both theoretical and, therefore, indirectly, applied computer science.

In this brief paper we can only make some first halting steps toward a general ontology theory. The bulk of this paper will be to argue for, and lay out in varying degrees of detail, a formal framework with the representational horsepower adequate for a robust ontology theory.

2 Intuitions

I begin with some intuitions to motivate the design of a framework for ontology theory.

1. Ontologies consist of propositions.
2. Propositions are not sentences, they are what sentences *express*; different sentences in different languages (or possibly the same language) can express the same proposition.
3. Propositions can be equivalent without being identical.
4. Propositions and ontologies are objects, things we can talk about.
5. The content of an ontology consists of the propositions involving concepts in the ontology that are entailed by the constituent propositions of the ontology.
6. Ontologies are comparable in terms of their content. In particular, two ontologies are equivalent if they have the same content.

3 Desiderata

In developing a general ontology theory our concern is describe the phenomenon, just as in the development of number theory or real analysis or, for that matter, computability theory. We therefore place no computational restrictions on expressiveness, and hence will avail ourselves of at least full first-order logic.[3]

However, we will need quite a lot more than that to satisfy the intuitions in the preceding section. Notably:

- Re (1) above, we need formal notions of *ontology* and *proposition*, and a notion of the relation between ontologies and the propositions they consist of.
- Re (2), we need a notion of proposition that is independent of any particular language.

[1] Department of Philosophy, Texas A&M University, College Station, TX 77840, email: cmenzel@tamu.edu

[2] Ironically, the very foundational work that began with the arithmetization of analysis and led to the development of mathematical logic ultimately resurrected the notion of an infinitesimal and an alternative foundation for analysis built thereon — so-called "nonstandard" analysis. See, e.g., [4] Chapter 3.

[3] As with both number theory and analysis, of course, we may want to explore computationally more tractable subtheories of our theory.

- Re (3), we need a notion of proposition robust enough to allow for distinct logically equivalent propositions.
- Re (4), we need to be able to name and quantify over propositions and ontologies; i.e., ontologies and propositions must be "first-class citizens" in ontology theory.
- Re (5), we need to be able to represent the notion of content, and hence (i) a notion of entailment that can hold between ontologies and propositions and (ii) a notion of the concepts within an ontology.
- Re (6), we need to be able to define notions of comparability in terms of ontological content.

I will satisfy these desiderata by developing a first-order theory of structured relations, of which propositions will be one species. Ontologies will be identified with 1-place relations, which for most purposes can play the role of classes. This theory will satisfy desiderata (1), (2), (3), and (4). By "structured" I mean that, although they will not be identified with formulas, relations will have a decomposable logical form similar to formulas. Together with a primitive modality, the structured nature of relations in turn will enable us to define a notion of entailment for propositions that will enable us to define a notion of content for ontologies, and hence to satisfy desiderata (5) and (6).

4 A Formal Framework for Ontology Theory

In this section I will define a language with appropriate expressive power for ontology theory and a corresponding semantics.

4.1 Syntax

To accomodate the narrow columns of the ECAI 2-column format, as much as anything, I will simply use the basic apparatus of standard *Principia Mathematica*-style first-order language, augmented with a number of useful constructs. I will call the language "$\mathcal{L}$".

Note that the unfriendliness of such languages in regard to computer processability is no more to the point here than it is with respect to group theory or computability theory. Our goal is theoretical — a mathematical theory of ontologies. Such work, of course, if sound, should lead to developments wherein computer processable languages are critical, but at this point processability is not an issue.

4.1.1 Lexicon

The lexicon consists of a countable set of *individual constants*, a denumerable set of *individual variables*, for each $n \geq 0$, a countable set of *n-place predicate constants* and a denumerable set of *n-place predicate variables* (jointly called n-place predicates), the reserved logical symbols $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$, $\forall$, $\exists$, λ, and $\Box$, and parentheses and brackets. Individual variables will consist of lower case letters, typically x, y, z, possibly with numerical subscripts. n-place predicate variables will consist of upper case letters with numerical superscripts (suppressed where context serves to indicate the arity of an n-place predicate), typically F^n, G^n, and H^n, possibly also with numerical subscripts. For purposes here, constants will consist of alphanumeric strings — other than the single-character strings already in use for the variables — beginning with an upper or lower case letter; dashes are also permitted to join alphanumeric strings. Typically, I will use a strings beginning with a lower case letter for constants that are intended to denote individual objects and strings beginning with an upper case letter for constants intended to denote relations.

4.1.2 Grammar

We define formulas and terms by a simultaneous recursion:

1. Any constant or variable is a *term*.
2. If π is an n-place predicate and $\tau_1, \ldots, \tau_n$ are any terms, $n \geq 0$, then $\pi(\tau_1, \ldots, \tau_n)$ is an (*atomic*) *formula* of $\mathcal{L}$. π is said to occur in *predicate position*, and each τ_i in *argument position*, in $\pi(\tau_1, \ldots, \tau_n)$. In the case where $n = 0$, we omit the empty parentheses and say that π standing alone is an atomic formula.
3. If φ, ψ are formulas, so are $\neg\varphi$, $\Box\varphi$, and $(\varphi \rightarrow \psi)$.
4. If φ is any formula and $\nu_1, \ldots, \nu_n$ any variables, then $(\forall\nu_1 \ldots \nu_n)\varphi$ is a formula.
5. If φ is a formula containing no occurrences of $\Box$, no bound variables occurring in predicate position, and no bound predicate variables, and $\nu_1, \ldots, \nu_n$ are any individual variables that do not occur free in any term occurring in φ, then $[\lambda\nu_1 \ldots \nu_n\ \varphi]$ is a predicable term.
6. Nothing else is a term or formula of $\mathcal{L}$

The usual definitions of $\wedge$, $\vee$, $\leftrightarrow$, and $\exists$ will be assumed.

There are two particularly distinctive features of $\mathcal{L}$. First, although the language of $\mathcal{L}$ contains so-called "higher-order" variables, unlike standard higher-order languages, these variables, and n-place predicates generally, are considered terms; they can occur as arguments to other predicates. Semantically speaking, as we will see explicitly below, this means that our universe is *type-free* — everything is an object; the quantifiers of the language will range over everything alike. Note this does *not* mean that there is no distinction between *kinds* of things. Notably, as noted already, our basic ontology includes relations as well as ordinary individuals. Rather, in accordance with intuition (4), it simply means that all of these things are in the universe of discourse, i.e., the range of the quantifiers. All entities — individuals, propositions, properties, and relations alike — are first-class logical citizens that jointly constitute a single domain of quantification. As such, properties and relations can themselves have properties, stand in relations, and serve as potential objects of reference.

Perhaps the strongest linguistic evidence for type freedom is the phenomenon of nominalization, whereby any verb phrase can be transformed into a noun phrase of one sort or another, most commonly, a gerund. So, for example, the verb phrase 'is famous' indicates a property that can be predicated of individuals, as in 'Quentin is famous'. Its gerundive counterpart, however, 'being famous', serves to denote a subject of further predication, as in, e.g., 'Being famous is all Quentin thinks about'. Intuitively, the verb phrase indicating the property predicated and the gerund indicating the object of Quentin's thoughts (i.e., the object possessing the property of being thought by Quentin) are the very same thing, the property of being famous.

In $\mathcal{L}$, this "dual role" of properties and relations — thing predicated vs. object of predication — is reflected in the fact that the same constant can play both traditional syntactic roles of predicate symbol and individual constant. Thus, in $\mathcal{L}$, we can write both

(1) $Famous(quentin)$

and

(2) $(\forall F)(ThinksAbout(quentin, F) \leftrightarrow (F = Famous))$

($\mathcal{L}$ retains no representation of the grammatical distinction between verb phrases — e.g., 'is famous' — and their gerundive counterparts

— e.g., 'being famous'. One could be added easily enough, of course, but as there is no semantic difference between verb phrases and their gerunds on a type free conception, any such representation would be semantically otiose.)

Because all objects are of the same logical type, it follows that any property can be predicated of any property and, in particular, a property can be predicated of (and, indeed, can exemplify) itself. Again, this comports with natural language; the property of being a property, for instance, is a property, and hence exemplifies itself. This is naturally represented in $\mathcal{L}$ in the obvious way:

(3) $Property(Property)$

It must be emphasized that the fact that we will be quantifying over properties, propositions, and relations generally does *not* in and of itself mean that $\mathcal{L}$ is higher-order. For that, one's semantics must involve higher-order quantifiers whose range includes a power set construction of some ilk over a domain of logical individuals. In our semantics, there is no such construction; there is but a single domain over which a single type of quantifier ranges.

The second distinctive feature of $\mathcal{L}$, and arguably the most prominent, is the presence of complex terms $[\lambda\nu_1 \ldots \nu_n\ \varphi]$. Intuitively, these terms denote complex relations. For instance, the term

(4) $[\lambda x\ Enjoys(x, salmon) \wedge Prefers(x, red\_wine, white\_wine)]$

indicates the property of enjoying salmon and preferring red wine to white. Terms with no bound λ-variables indicate 0-place relations, i.e., propositions. In this case the λ can be dropped. Thus,

(5) $[\forall x(Planet(x) \rightarrow Larger(sun, x))]$

indicates the proposition that the sun is larger than all of the planets. This feature of $\mathcal{L}$ is particularly important, as ontologies in the proposed theory will be characterized roughly as classes of propositions, and the logical connections between ontologies will be expressed in terms of logical relations between propositions. λ-terms enable us to talk about the propositions in a given ontology explicitly. And as we will see, they are also extremely useful for defining a variety of important auxiliary notions.

4.1.3 On Syntactic Restrictions on Term Formation

Clause (5) in the grammar for $\mathcal{L}$ imposes a number of restrictions on the formation of complex terms. The most noteworthy of these is the restriction permitting only individual variables to be bound by the λ operator in complex terms. This restriction avoids the Russell paradox, as without that restriction the term $[\lambda F\ \neg F(F)]$ — indicating, intuitively, the property of non-self-exemplification — would be legitimate. The grammar would then permit the construction of the atomic formula $[\lambda F\ \neg F(F)]([\lambda F\ \neg F(F)])$, which, by the logical principle of λ-conversion ((10) below), could be proven equivalent to its negation. However, the restriction that prevents the paradox is not *ad hoc*. Its justification — which will become clear in Section 4.2 — is that there is simply no intuitive logical operation that yields relations whose logical form corresponds to such terms, and hence no warrant for permitting them. The avoidance of Russell's paradox falls out as a consequence of this restriction, and hence is *explained* rather than merely avoided: the paradox arises from a theoretically unwarranted assumption about the structure of complex relations, much as the corresponding paradox of self-membership arises from a theoretically unwarranted assumption about the nature and structure of sets (see, e.g., [2]).

Clause (5) imposes a number of other restrictions on the formation of terms that are, in fact, dispensable in the sense that we could in fact provide a reasonable semantics for them. Specifically:

- The requirement that λ-bound variables all occur free in φ rules out such terms as $[\lambda xy\ Px]$ that contain vacuous λ-bound variables;[4]
- The restriction on free occurrences of λ-bound variables within complex terms occurring in φ rules out such terms as $[\lambda xy\ P[\lambda zQxz]y]$;[5]
- The restriction on bound occurrences of predicate variables within complex terms occurring in φ rules out such terms as $[\lambda xy\ (\exists F^1)y = F^1]$;[6]

However, the terms that would be permitted without these restrictions are inessential to our purposes here and hence allowing them would introduce unnecessary technical complexity.

While the restrictions to non-modal formulas in the formation of terms is, like the two above, also inessential, it has a certain intuitive warrant. For, unlike the three restrictions above, this restriction reflects an important feature of the intended domain guiding the development of the current framework. Specifically, we are formulating a theory of *first-order* ontologies, that is, ontologies whose constituent propositions are expressible by sentences in a non-modal first-order language (hence in any weaker sublanguage thereof). This is, of course, not to say that there are no modal (or higher-order) ontologies. However, the vast majority of existing ontologies are first-order, and it seems quite unlikely that this will change with the development of the Semantic Web if the expressiveness of its basic language is to be on the order of DAML+OIL. Therefore, to provide the capacity to express modal propositions, at this point, seems unwarranted.

A *theory* of ontologies, however, does need this expressive power. Specifically, modality is useful for characterizing the nature of ontologies and their logical connections. Most notably, perhaps, as will be seen explicitly in Section 7 below, the modal component of the language of our theory enables us to define a robust notion of entailment which, in turn, can be used to formulate a correspondingly robust notion of ontological content.

4.2 Semantics

In this section I will build upon work by Bealer [1], Zalta [7], and Menzel [5] to develop a rich "meta-ontology" of structured relations.

4.2.1 Model Structures

A *model structure* $\mathfrak{M}$ for $\mathcal{L}$ is a 5-tuple $\langle D, W, dom, Op, ext\rangle$. Here $D = \bigcup\{A, R\}$ is the *domain* of $\mathfrak{M}$, and consists of the union of two mutually disjoint sets A and R. A is the set of *individuals* of D and R is the set of *relations*, of which we consider *propositions* a species. R itself can be partitioned in two significant ways. First, R is the union of two mutually disjoint nonempty sets R^p and R^c, intuitively, the sets of logically primitive and logically complex relations, respectively. Additionally, R is the union of denumerably many nonempty sets $R_0, R_1, \ldots$, each R_n being, intuitively, the class of n-place relations. We let R^p_n and R^c_n be $R^p \cap R_n$ and $R^c \cap R_n$, respectively. W

[4] Such terms are easily accommodated by means of a further set of logical operations Vac_i that insert vacuous argument places into the i^{th} "slot" in the argument structure of a relation.

[5] See Menzel [5] for an account of the semantics of such terms and surrounding philosophical issues.

[6] Again, see [5] for the semantics of such terms.

is a nonempty set, intuitively, a set of "possible worlds" or "possible situations." More formally, W provides us with a model of modality that enables us to represent entailment and other logical relations between ontologies. Accordingly, dom is a function that maps every element w of W to a subset $dom(w)$ of D representing, intuitively, the set of things that "exist" in the possible world w.

The next element of a model structure, Op is a set of five sets of logical operations: a set of *predication* operations, $\{Pred^n_{i_1 \ldots i_k} : 0 < i_1 < \ldots < i_k \leq n; k \geq 0$[7]$\}$; a set of two *boolean* operations, $\{Neg, Impl\}$; a set of *universalization* operations, $\{Univ_{i_1 \ldots i_k} : 1 \leq i_1 < \ldots < i_k \leq n;\}$; a set of *conversion* operations, $\{Conv^i_j : 1 \leq i < j < \omega\}$; and a set of *reflection* operations, $\{Refl^i_j : 1 \leq i < j < \omega\}$. These operations "construct" logically complex relations from individuals and less complex relations in D. Specifically, for all n:

- $Pred^n_{i_1 \ldots i_k} : R_n \times D^k \longrightarrow R^c_{n-k}$ $(1 \leq i_1 < \ldots < i_k \leq n; k \geq 1)$;
- $Neg : R_n \longrightarrow R^c_n$;
- $Impl : R_n \times R_m \longrightarrow R^c_{n+m}$;
- $Univ_{i_1 \ldots i_k} : R_n \longrightarrow R^c_{n-k}$, for $k \leq n$.
- $Conv^i_j : R_n \longrightarrow R^c_n$ $(1 \leq i < j \leq n)$;
- $Refl^i_j : R_n \longrightarrow R^c_{n-1}$ $(1 \leq i < j \leq n)$;

We stipulate that $Pred^n$ (i.e., $Pred^n_{i_1 \ldots i_k}$ for $k = 0$) is just the identity relation on R^n,[8] and that R^c is just the union of the ranges of the logical operations, i.e., $R^c = \{\bigcup Range(f) : f \in \bigcup Op\}$. To capture fine-grainedness, it is assumed that all of these operations are one-to-one and that the ranges of all of the operations are pairwise disjoint — similar to their syntactic counterparts, the "logical forms" of relations formed from these operations are all distinct from one another.

Finally, let D^n be the set of all n-tuples over D and let $D^* = \bigcup_{0<n<\omega} D^n$. ext is a function on $R \times W$ such that for all $r \in R_n$, $w \in W$, $ext(r, w) \subseteq D^n$. Note that, for $r \in R^0$, only two extensions are possible: $\{\langle\rangle\}$, i.e., D^0 itself, and the empty set $\varnothing$. In this case it is useful to think of the former as the truth value $\top$ (truth) and the latter as the truth value $\bot$ (falsity).

The behavior of ext is constrained further by the logical operations in $\bigcup Op$. Some notational conventions will be helpful for stating these constraints. For $A \subseteq D^n$, let $\overline{A}$ be $D^n - A$. Where $s \frown s'$ is the concatenation of two sequences (tuples) s, s', for subsets A, B of D^n and D^m, respectively, let $A \frown B = \{a \frown b : a \in A, b \in B\}$. Where $1 \leq i_1 < \ldots < i_j \leq n$, we let $\langle b_1, \ldots, b_n\rangle^{i_1 \ldots i_j}_{a_1 \ldots a_j}$ be result of replacing each b_{i_k} with a_k, and we let $\langle b_1, \ldots, b_n\rangle^{i_1 \ldots i_j}$ be the result of deleting each b_{i_k} from $\langle b_1, \ldots, b_n\rangle$. Given this, let $r \in R_n$, $q \in R_m$; then:

- $ext(Pred^n_{i_1 \ldots i_k}(r, a_1, \ldots, a_k), w) =$
  $\{\langle b_1, \ldots, b_n\rangle^{i_1 \ldots i_k} : \langle b_1, \ldots, b_n\rangle^{i_1 \ldots i_k}_{a_1 \ldots a_k} \in ext(r, w)\}$;[9]
- $ext(Neg(r), w) = \overline{ext(r, w)}$;
- $ext(Impl(q, r), w) = \overline{ext(q, w)} \frown D^n \cup D^m \frown ext(r, w)$
- $ext(Univ_{i_1, \ldots, i_j}(r), w) =$
  $\{\langle a_1, \ldots, a_n\rangle^{i_1 \ldots i_j} : \forall b_1, \ldots, b_j \in dom(w),$
  $\langle a_1, \ldots, a_n\rangle^{i_1 \ldots i_j}_{b_1 \ldots b_j} \in ext(r, w)\}$
- $ext(Conv^i_j(r), w) =$
  $\{\langle a_1, \ldots, a_{i-1}, a_j, \ldots, a_{j-1}, a_i, \ldots, a_n\rangle : \langle a_1, \ldots, a_n\rangle \in ext(r, w)\}$
- $ext(Refl^i_j(r), w) =$
  $\{\langle a_1, \ldots, a_{j-1}, a_{j+1}, \ldots, a_n\rangle : \langle a_1, \ldots, a_n\rangle \in ext(r, w)$ and $a_i = a_j\}$

Constituency and Logical Form The intuitive picture here is a "quasi-constructive" one similar to the intuitive picture that underlies the iterative conception of sets. We begin with a set A of individuals and a set R^p of logically simple relations. The logically simple relations are thought of as the meanings of the primitive predicates in an ontology. The predication functions applied to primitive relations and individuals yield basic atomic relations — notably, basic atomic propositions — and the remaining logical operations applied to these yield logically complex relations. These in turn, can be arguments to further applications of the logical operations, yielding an "iterative hierarchy" of relations of increasing complexity. Intuitively, then, relations in R are either primitive or are "built up" from individuals and other relations via the logical operations, and the manner in which a relation is so built up can be thought of as its *logical form*. So, for example, our example proposition (5) — $[\forall x(Planet(x) \rightarrow Larger(sun, x))]$ — that the sun is larger than every planet would be built up from the property of being a planet, the 2-place relation of being larger than, and the sun as follows. $Pred^2_1$ applied to the larger-than relation and the sun yields the property

(6) $[\lambda y\ Larger(sun, y)]$

of being something that the sun is larger than.[10] The boolean "material implication" operator $Impl$ applied to the property of being a planet and (6) yields the relation

(7) $[\lambda xy\ Planet(x) \rightarrow Larger(sun, y)]$

that a bears to b just in case a is not a planet or the sun is larger than b. The reflection operation $Refl^1_2$ applied to (7) "collapses" its two argument places into one to yield the property

(8) $[\lambda x\ Planet(x) \rightarrow Larger(sun, x)]$

of being something such that if it is a planet, then the sun is larger than it. Finally, application of the "quantification" operator $Univ_1$ yields our desired proposition (5). In a single equation, then, we have

(9) $(5) = Univ_1(Refl^1_2(Impl(Planet, Pred^2_1(Larger, sun))))$.

The manner in which a relation is built up from individuals and other relations can be thought of as its logical form. We can make this idea rigorous as follows. Say that a *constituency tree* for an element $r \in R$ is any labeled ordered tree T whose nodes are in D and whose root node is r, such that, for every node e of T, the daughter nodes $e_1, \ldots, e_j$ of e are such that, for some operation $F \in \bigcup Op$, $F(e_1, \ldots, e_j) = e$. A constituency tree T for r is *complete* iff every leaf node o of T is an individual or a primitive relation, i.e., iff $o \in A \cup R^p$. Given the constraints on our logical operations it is easy to show that every $r \in R$ has exactly one complete constituency tree, which we can therefore identify with the *logical form* of r. We define an object $o \in D$ to be a *constituent of* a relation $r \in R$ just in case

[7] If $k = 0$, then $i_1 \ldots i_k$ is the null sequence, which we want to allow here.

[8] This stipulation will yield as logical truths all instances of $\pi = [\lambda\nu_1 \ldots \nu_n\ \pi(\nu_1, \ldots, \nu_n)]$, for all n-place predicates π.

[9] Note that by the definition of the *Pred* functions, we always have $k \leq n$ in $Pred^n_{i_1 \ldots i_k}$.

[10] I am of course *using* the term '$[\lambda y\ Larger(sun, y)]$' here, not mentioning it; I am not talking about the term itself, but rather the property it intuitively denotes under the standard English meanings of the constituent constants.

e is a node in the complete constituency tree for r. o is a *primitive* constituent of r iff o is a constituent of r and $o \in A \cup R^p$. The notion of constituency will be important for defining the concept of ontological content in Section 7 below.

4.2.2 Denotations, Interpretations, and Truth

Denotations for the terms of $\mathcal{L}$ relative to a model structure $\mathfrak{M}$ are determined by partitioning the class of complex terms according to their syntactic form. In brief, where τ is $[\lambda\nu_1 \ldots \nu_n\ \varphi]$, if the order of the λ-bound variables in φ does not correspond to $\nu_1, \ldots, \nu_n$, then τ is the conversion^i_j of an appropriate term τ'. Otherwise, if one of the λ-bound variables occurs free more than once in φ, then τ is a reflection^i_j of an appropriate τ'. Otherwise, τ is classified as the universalization$_{i_1,\ldots,i_j}$, implication, negation, or predication$^n_{i_1,\ldots,i_k}$ of the appropriate sort depending on the logical form of φ. Complex terms of the form $[\lambda\nu_1 \ldots \nu_n\ \pi(\nu_1, \ldots, \nu_n)]$, for any predicate π — i.e., those of the form predication^n — are said to be *trivial*, as they indicate no more logical complexity than the constitutive predicate π.

Given a model structure $\mathfrak{M}$, let d be a function assigning elements of the domain D of $\mathfrak{M}$ to the individual constants and variables of $\mathcal{L}$ and elements of R_n to the n-place predicates of $\mathcal{L}$. Such a d is known as a *denotation function* for $\mathcal{L}$ relative to $\mathfrak{M}$. Denotations for complex terms are then assigned by extending d in an obvious way that exploits the close parallel between the syntactic form of complex terms and the logical forms of complex relations:

- If τ is the conversion^i_j of τ', then $d(\tau) = Conv^i_j(d(\tau'))$.
- If τ is the reflection^i_j of τ', then $d(\tau) = Refl^i_j(d(\tau'))$.
- If τ is the universalization$_{i_1,\ldots,i_j}$ of τ, then $d(\tau) = Univ_{i_1,\ldots,i_j}(d(\tau'))$.
- If τ is the implication of τ' and τ'', then $d(\tau) = Impl(d(\tau'), d(\tau''))$.
- If τ is the negation of τ', then $d(\tau) = Neg(d(\tau'))$.
- If τ is the predication$^n_{i_1,\ldots,i_j}$ of τ' of $\tau_1, \ldots, \tau_j$, then $d(\tau) = Pred^n_{i_1,\ldots,i_j}(d(\tau'), d(\tau_1), \ldots, d(\tau_j))$.

We say that a denotation function d' for $\mathcal{L}$ relative to $\mathfrak{M}$ is a *ν-variant of d*, for any variable ν, just in case, for all variables $\mu \neq \nu$, $d'(\mu) = d(\mu)$.[11]

An *interpretation* $\mathfrak{A}$ of $\mathcal{L}$ is a pair $\langle \mathfrak{M}, d\rangle$ consisting of a model structure $\mathfrak{M} = \langle D, W, dom, Op, ext\rangle$ and a denotation function d for $\mathcal{L}$ relative to $\mathfrak{M}$. For any variable ν, a *ν-variant of* $\mathfrak{A} = \langle \mathfrak{M}, d\rangle$ is any interpretation $\mathfrak{A}' = \langle \mathfrak{M}, d'\rangle$ such that d' is a ν-variant of d.

Let $\mathfrak{A} = \langle \mathfrak{M}, d\rangle$ be an interpretation, where $\mathfrak{M} = \langle D, W, dom, Op, ext\rangle$. *Truth* at a world $w \in W$ in $\mathfrak{A}$ for the formulas of $\mathcal{L}$ is defined in the standard sort of way:

- $\pi(\tau_1, \ldots, \tau_n)$ is true at w in $\mathfrak{A}$ iff $\langle d(\tau_1), \ldots, d(\tau_n)\rangle \in ext(d(\pi), w)$.
- $\neg\varphi$ is true at w in $\mathfrak{A}$ iff φ isn't.
- $(\varphi \rightarrow \psi)$ is true at w in $\mathfrak{A}$ iff either φ isn't or ψ is.
- $\forall\nu\varphi$ is true at w in $\mathfrak{A}$ iff φ is true at w in all ν-variants of $\mathfrak{A}$.
- $\Box\varphi$ is true at w in $\mathfrak{A}$ iff φ is true at w' in $\mathfrak{A}$, for all $w' \in W$.

5 Proof Theory

The proof theory for this semantics is an extension of classical first-order logic with identity. Notably, there are principles of identify for complex terms that ensure fine-grainedness, e.g., that no universalization is an implication or a negation, that predications are identical iff they are predications of the same relation of exactly the same objects, and so on.

More relevant for ontology theory, however, is a principle of λ-conversion that takes as axioms all instances of:

(10) $[\lambda\nu_1 \ldots \nu_n\ \varphi](\tau_1, \ldots, \tau_n) \leftrightarrow \varphi^{\nu_1,\ldots,\nu_n}_{\tau_1,\ldots,\tau_n}$,

where $\varphi^{\nu_1,\ldots,\nu_n}_{\tau_1,\ldots,\tau_n}$ is the result of replacing every free occurrence of ν_i in φ with τ_i. This principle lets us move freely between statements about individuals and the attribution of complex properties and relations to those individuals, e.g.,

(11) $[\lambda x\ Enjoys(x, salmon) \wedge Prefers(x, wine, beer)](jo)$
$\leftrightarrow Enjoys(jo, salmon) \wedge Prefers(jo, wine, beer)$

Notably, as we will see below, this principle will give us the ability to move from *talking about* the propositions in an ontology to *using* them in logical inferences. Like the other axioms of our theory, (10) is easily shown to be valid relative to the semantics above.

6 The Logic of Constituency

The notion of constituency enables us to capture the intuitive fact that different ontologies contain different concepts: the concepts in an ontology are simply the properties and relations that are constituents of the propositions of that ontology. Our fine-grained, structured notion of properties, relations, and propositions gives us a rigorous foundation for analyzing and exploiting the notion of constituency. We have characterized constituency model theoretically above in Section 4.2.1. In this section we capture the notion axiomatically. We begin with a schema:[12]

(12) $Const(\tau, \tau')$, where τ' is a nontrivial complex term and τ occurs free in τ'

That is, any term occurring free within a complex term indicates a constituent of the relation denoted by the complex term.

Next, we note that the constituency relation is a strict partial ordering, i.e., it is transitive and asymmetric (hence also irreflexive):

(13) $(Const(p, q) \wedge Const(q, r)) \rightarrow Const(p, r)$

(14) $Const(q, r) \rightarrow \neg Const(r, q)$

Finally, we can define an object to be primitive just in case it has no constituents:

(15) $Prim(x) =_{df} \neg(\exists q)\, Const(q, x)$

This reflects the model theoretic fact that the ranges of the logical operations (other than the "trivial" $Pred^n$ operations) are all subsets of R^c.

7 Content

As indicated, content is best cashed out in terms of some notion of entailment. In classical first-order logic, entailment is usually understood model theoretically. Ciociou and Nau [3] have taken some steps in this direction in developing a formal notion of intertranslatability between ontologies. For them, ontologies are understood as sets of

[11] Thus, as it is often informally put, d' differs from d at most in what it assigns to μ.

[12] Recall that a trivial complex term is of the form $[\lambda\nu_1 \ldots \nu_n\ \pi(\nu_1, \ldots, \nu_n)]$.

sentences, and the content of an ontology is understood in terms of its formal models: the content of an ontology O consists in the set of its semantic consequences, i.e., the set of sentences that are true in all the models of O. This approach thus can yield a robust notion of common ontological content across different languages in terms of shared models.

This approach is clear and insightful, but suffers from two shortcomings. First, though a notion of common content is possible on this approach, the notion of ontology is still language-dependent; an ontology is a set of sentences in some language. This violates intuitions 1 and 2 above, which jointly imply that ontologies are classes of language-independent propositions. More seriously, however, the approach — as a basis for a general theory of ontologies — is unwieldy. Content is understood in terms of the models of a theory. Hence, on this approach, one has to import the full apparatus of first-order model theory — basic set theory, formal languages, interpretations, model theoretic truth and entailment, etc — just to define a reasonable notion of ontological content. Moreover, the model theoretic approach makes for a rather austere and formal notion of content — to identify the *meaning* of a sentence with a set of models is rather far removed from ordinary semantic notions.

Tbough the present approach has a strong model theoretic component, that component serves only to ground a first-order theory of ontologies and their content; no linguistic or model theoretic entities, properties, or relations are introduced into the theory.[13] Rather, it develops an account of ontologies and their content that is language independent and grounded in the intuitive notion of propositions — rather than the austere and abstract notion of a model — as the basic semantical unit of meaning.

To get at the relevant notion of entailment in our theory, recall once again that, intuitively, ontologies can be thought of as classes of propositions. The notion of a proposition easily defined in terms of our "higher-order" quantifiers:

(16) $Proposition(p) =_{df} (\exists F^0)p = F^0$

Understanding classes as properties, we can now define an ontology to be a nonempty class of propositions:

(17) $Ont(O) \quad =_{df} \quad (\exists F^1)O = F^1 \wedge (\exists x)F^1(x) \wedge \forall p(F^1(p) \rightarrow Proposition(p))$

The notion of entailment we are after involves both modality and our notion of constituency. We first define a constituent of an ontology O to be a constituent of one of the propositions in O:

(18) $OntConst(x, O) =_{df} Ont(O) \wedge (\exists p)(O(p) \wedge Const(x, p))$

Next, say that an ontology O entails a proposition F^0 just in case F^0 must be true if all the propositions in O are true:

(19) $Entails(O, F^0) \quad =_{df} \quad Ont(O) \wedge \Box((\forall G^0)(O(G^0) \rightarrow G^0) \rightarrow F^0)$

Now say that F^0 and O *share primitives* if every primitive constituent of F^0 is a constituent of O:

(20) $ShPrim(F^0, O) =_{df} Ont(O) \wedge (\forall x)((Prim(x) \wedge Const(x, F^0)) \rightarrow OntConst(x, O))$

[13] Though of course we define a model theory for the *language* $\mathcal{L}$ of our theory, but that's just a matter of our own metatheoretic housekeeping: it simply provides a proper theoretical foundation for the language we are using to express our theory; the model theory for $\mathcal{L}$ is not itself a part of ontology theory.

Thus, combining (19) and (20), we have the notion of entailment we are after:

(21) $StrEntails(O, F^0) =_{df} Entails(O, F^0) \wedge ShPrim(O, F^0).$

That is, an ontology O strongly entails a proposition F^0 just in case O entails F^0 and F^0 and O share primitives; that is, intuitively, if O entails F^0 and F^0 is "built up" from the same pool of concepts and objects — the same "conceptual vocabulary" — as the propositions in O. We will sometimes write "$O \Rightarrow F^0$" for "$StrEntails(O, F^0)$".

The content of an ontology, then, can be thought of as all of the propositions that it strongly entails. As it happens, we cannot strictly *define* the content of an ontology as an object. However, for theoretical purposes, strong entailment appears to be all we need. For example, we can say that one ontology O *subsumes* another O' just in case the content of O' is included in that of O, i.e., just in case O strongly entails every proposition that O' does:

(22) $Subsumes(O, O') \quad =_{df} \quad Ont(O) \wedge Ont(O') \wedge (\forall p)(O' \Rightarrow p \rightarrow O' \Rightarrow p)$

Ontologies can then be said to be *equivalent* just in case they subsume each other:

(23) $Equiv(O, O') =_{df} Subsumes(O, O') \wedge Subsumes(O', O).$

Subtler metrics for comparison are of course also possible, e.g., two non-equivalent ontologies might nonetheless share all or some of their primitives. More generally, the notions defined above provide a rich framework for analyzing a wide variety of notions relevant to understanding the nature of, and logical relations between, ontologies.

8 Integration

A major extension of this work that goes far beyond the current scope will consist in developing a theory of integration. At a purely abstract level, integration is fairly straightforward. One can import several ontologies into the language $\mathcal{L}$ of our approach by creating a separate namespace for the terms in each ontology and translating them from the language of the ontology into $\mathcal{L}$. There will thus be, initially, no possibility of name conflicts. Because the principle $[\varphi] \leftrightarrow \varphi$ is valid, it will be possible to move seamlessly back and forth between *using* the axioms of a given ontology to investigate its properties and talking more generally about the ontology and its content.

Because distinct ontologies are imported with separate namespaces, there is no danger of logical inconsistency arising from incompatible ontologies. Integration can proceed by identifying or otherwise logically connecting the concepts (objects, properties, relations, and propositions) expressed across ontologies. Thus, for instance, it might be postulated that two concepts (properties) from different ontologies are identical; or that one concept subsumes the other; or that for every instance a of one there are two instances of the other that bear some relation to a; and so on. In this way the logical connections between ontologies can be mapped clearly and rigorously and with ever greater precision.

However, while this account of integration is theoretically adequate as far as it goes, a complete treatment will have to include a theory of languages that connects sets of sentences with the ontologies that they express, and which should lead to more practical applications of the theory. Investigating integration at this more applied level will be the next phase of this project.

9 Conclusion

It will be possible for ontology to make significant progress toward the lofty goals workers in the area are pursuing only if it has proper theoretical foundations. For such goals can be reached only if there is a clear, generally shared understanding of the subject matter of ontology, one that makes it possible clearly to define the scope of the discipline, to identify its subject matter, and chart a course toward the resolution of its outstanding problems. The approach in this paper shows promise for providing these essential theoretical underpinnings

REFERENCES

[1] Bealer, G., *Quality and Concept*, Clarendon Press, 1982.

[2] Boolos, G., "The Iterative Conception of Set," *Journal of Philosophy* 68, 215-231.

[3] Ciocoiu, M. and Nau, D. "Ontology-Based Semantics," in Cohn, A., Giunchiglia, F., and Selman, B., (eds.), *Principles of Knowledge Representation and Reasoning. Proceedings of the Seventh International Conference*, 539-546, Morgan Kaufmann.

[4] Enderton, H. B., *A Mathematical Introduction to Logic*, Academic Press, 1972.

[5] "The Proper Treatment of Predication in Fine-grained Intensional Logic," in J. Tomberlin (ed.), *Philosophical Perspectives, vol. 7: Language and Logic*, 1993, Ridgeview Publishing Co., 1993, 61-87.

[6] Turner, R., 1992, Properties, Propositions, and Semantic Theory, in M. Rosner and M. Johnson (eds.), *Computational Linguistics and Formal Semantics*, Studies in Natural Language Processing, pp. 159-180, Cambridge University Press.

[7] Zalta, E., *Abstract Objects: An Introduction to Axiomatic Metaphysics*, Dordrecht: D. Reidel, 1983

Resolving Terminological Heterogeneity In Ontologies

Prasenjit Mitra and Gio Wiederhold
Infolab, Stanford University,
Stanford, CA 94305, USA [1]
{mitra, gio}@db.stanford.edu

Abstract. A system that enables interoperation among information sources using ontologies needs to resolve the terminological differences between ontologies. In this work, we present several methods that we have designed to match terms used in different ontologies. We have implemented two methods based on linguistic similarities of terms used in the ontologies. The first looks up a dictionary or semantic network like WordNet and the second determines similarities of words based on word similarity compuoed from a domain-specific corpus of documents. We discuss our experiments that indicate that a method that uses both heuristics produces good results.

1 Introduction

Often, we cannot answer a query from a single source, and need to compose information from multiple information sources. These information sources are autonomously created and maintained. Integrating the information in them to create a single source is not an option where the owners of the information sources prefer to maintain their autonomy. The merging approach of creating an unified source is not scalable and is costly. Besides, an integrated information source would need to be updated as soon as any information in any individual source changes [11]. Furthermore, in certain cases a complete unification of a large number of widely disparate information sources into one monolithic information source is not feasible due to unresolvable inconsistencies between them that are irrelevant to the application. For a particular application, resolution of inconsistencies between a pair of knowledge sources is typically feasible, but it becomes nearly impossible when the objective is undefined and the number of sources is large.

Due to the complexity of achieving and maintaining global semantic integration, the merging approach is not scalable. We have adopted a distributed approach which allows the sources to be updated and maintained independent of each other and enables composition of information via interoperation.

1.1 The Need for Autonomous Ontologies

Ontologies are increasingly being used to assist the integration of information. They specify the terminology (and its semantics) used in information sources. These sources are autonomously created and maintained.

The alternative to individual ontologies for individual sources is to use standard ontologies across multiple information sources. Efforts to create and use standardized ontologies have met with limited success due to the different requirements of the different businesses that construct the information sources. Even if such efforts succeed in creating a standard ontology, the large size of such an ontology results in poor performance while using the ontology.

Everyday new discoveries expand our knowledge and change our views of the universe that we live in. Any ontology representing such knowledge has to be updated periodically. The maintainers of the information sources that use the standard ontology will have to agree on the updates being proposed and on the restructuring of the ontology. They may have entirely different applications in mind or may not subscribe to a newly discovered theory. Furthermore, some participants might see the changes required to support the proposed updates as an unnecessary imposition since restructuring the information source will require substantial effort on their part. Thus generating new consensus on updates to the standard ontology is a time-consuming and tenuous process. For quickly changing fields, arriving at a consensus within a short period of time is not even feasible.

Besides, even if a standard ontology is devised and widely used in future, the ontologies that exist today cannot be wished away. To handle such legacy ontologies, and to allow interoperation among information systems with autonomous ontologies, we need to interoperate among the ontologies themselves.

1.2 Resolving Semantic Heterogeneity

Problems of heterogeneity in hardware, operating systems, and data structures have been widely addressed, but issues of diverse semantics have been handled mainly in an ad-hoc fashion. While composing information from information sources, we need to ensure that the information that we are composing have some semantically meaningful relationship. Semantic heterogeneity among information sources needs to be resolved to enable meaningful information exchange or interoperation among them.

The two major sources of heterogeneity among the sources are as follows: First, different sources use different data formats and modeling languages to represent their data and meta-data. Second, sources using the same data format differ in their structure and semantics of the terminology they use. Such heterogeneity are a result of the autonomous nature of the ontologies and the fact that information sources are constructed by different people with different objectives in mind.

Often different sources use different terminologies to describe the objects in the sources. The same term, used in different sources, often have overlapping or somewhat different semantics, e.g., the term “nail” has entirely different semantics in a “cosmetics” ontology and the “carpentry” ontology. Similarly, different sources, often, use dif-

[1] This work has been supported by the AFOSR New World Vistas program and the DARPA DAML program.

ferent terms to refer to semantically similar objects, e.g., the terms "truck" and "lorry" in two transportation ontologies might refer to the same class of objects.

In order to enable interoperation, we intend to capture the semantic bridges between two ontologies using *articulation rules*. These rules express the relationship between two (or more) concepts belonging to the ontologies that we seek to interoperate. Since these ontologies can be fairly large, establishing such rules manually is a very expensive and laborious task. Fully automating the process is also not feasible. First, despite the rapid advances made in the field of natural language processing, the technology still remains inadequate to automatically resolve semantic heterogeneity among these information sources using different terminology. Second, even though ontologies expose some of the semantics of the terms and their relationships, they often remain incomplete or inadequate if we consider the needs of the various applications that use them.

The problem of ontology alignment has been studied for some time. Tools like OntoMorph [4], PROMPT [10], and Chimaera [8] help significantly automate the process. However, these tools do not contain a component that identifies concept names that are linguistically similar automatically and use that knowledge as the basis for furhter alignment of the ontologies. They require manual construction of articulation rules or base their matches on the structure of the ontologies. Our approach provides a greater degree of automation while keeping the option of a human expert to ratify the suggested articulation. A similar problem is that of schema matching in databases. However, most of the techniques used in matching tools [14],[7], [5], [9], [12], [3] etc. are not adequate when the primary differences among sources are due to differences in terminology in sources with little structural similarity or when instance data is not available and would provide poor results due to the absence of good structural similarity or absence of instance data in our applications.

In this paper, we propose a semi-automated algorithm for resolving the terminological heterogeneity among the ontologies and establishing the articulation rules necessary for meaningful interoperation. This algorithm forms the basis of the *articulation generator* for our ONtology compositION sytesm (ONION). Our experiments show that basing such matching on structural information is inadequate. We describe several heuristics to resolve the terminological heterogeneity among ontologies. Experimental results show that combining the information obtained by using multiple heuristics provides a better match between semantically related terms in the ontologies.

2 Ontologies and Their Articulations

In this work, we assume that the ontologies we use are represented as a graph along with a set of logical rules. Formally, an ontology $O = (G, R)$ is represented as a directed labeled graph G and a set of rules R. The graph $G = (V, E)$ comprises a finite set of nodes V and a finite set of edges E. The label of a node is given by a non-null string that is often a noun-phrase that represents a concept name. The label of an edge is the name of a semantic relationship among the concepts and can be null if the relationship is not known. The label of an edge can be any user-defined relationship. The set of relationships with pre-defined semantics is $\{SubClassOf, PartOf, AttributeOf, InstanceOf, ValueOf\}$. All other relationships are not interpreted by the articulation generator in ONION.

Articulation rules are of two types - ones that are simple statements of the form $(Match"DepartmentofDefence""DefenseMinistry")$ expressing matches between equivalent concepts and the more complex rules expressed in datalog that are mostly supplied by the expert. We use the relation $(MatchConcept1Concept2)$ to indicate that the two concepts $Concept1$ and $Concept2$ are related (above an acceptable threshold using an expert-supplied metric of relatedness that can vary from application to application). $Match$ does not indicate the exact semantic relationship between the two concepts, for example, whether they have a class-subclass relationship, or are equivalent etc. Therefore, $Match$ gives a coarse relatedness measure and it is upon the human expert to then refine it to something more semantic, if such refinement is required by the application. For example, the human expert might indicate that by default all concepts that $Match$ are to be taken to be equivalent unless otherwise noted by the expert.

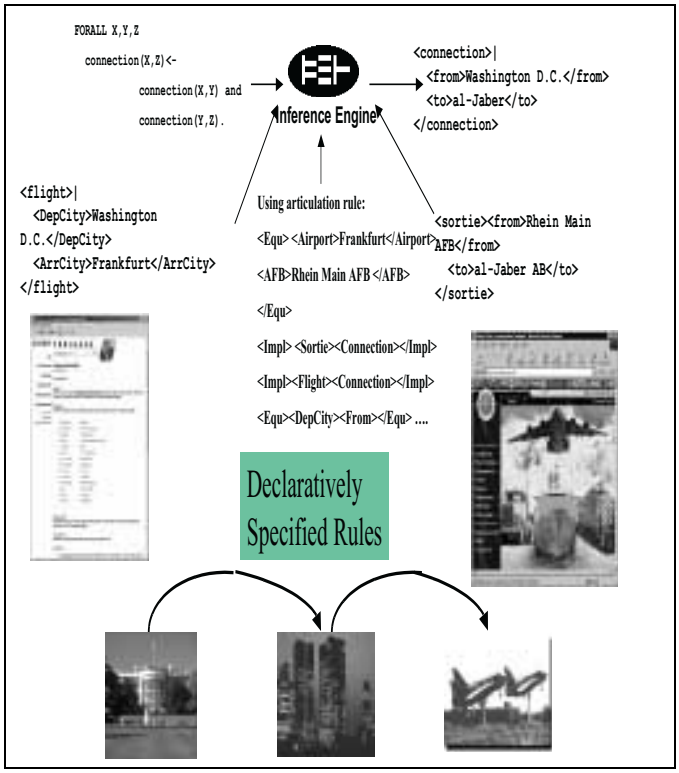

Figure 1. An application using an articulation between the United Airlines Ontology and the TRANSCOM Ontology

In Figure 1, we show an example articulation. On the left hand side, is a portion of the United Airlines Ontology and on the right a portion of the TRANSOM Ontology. These ontologies were constructed manually for experimentation. The objective of the application is to transport military men and materiel from Washington D.C. to Al Jabar Airbase in Kuwait. A combination of commercial flights and special purpose sorties is to be used to meet the transport objective.

The United Airlines source has $flight$, whose $DepCity$ is $Washington\ D.\ C.$ and $Arrity$ is $Frankfurt$. This corresponds to a flight from Washington D. C. to Frankfurt. There exists an articulation rule, supplied by the domain expert, that says that the connection relation is transitive.

The TRANSCOM source has a $sortie$ that runs from $RheinMainAFB$ in Frankfurt, Germany to $AlJabar$ airbase in

Kuwait.

We establish the articulation rules semi-automatically. They indicate that the $Frankfurt$ airport is the co-located with the $RheinMainAFB$. It tells us that if there is a $sortie$ or a $flight$ between two cities, then there is a $connection$ between them. It also indicates that $DepCity$ in the United ontology is the same as $From$ in the TRANSCOM ontology. Due to lack of space in the figure, the rule that states that $United.ArrCity$ is equivalent to $TRANSCOM.To$ is not shown.

Using these rules, an inference engine can easily establish that there is a connection between $Washington\ D.\ C.$ and $AlJabarAirbase, Kuwait$.

The tool generates and suggests the simpler articulation rules to indicate the terms in the two ontologies that are related. The expert then validates these suggestions and the final set of articulation rules are stored to be used during query rewriting and execution.

3 Generation of Ontology Articulations

ONION has an automated articulation generator (ArtGen) that suggests articulations based on a library of heuristic matchers. Each matcher matches terms in the two ontologies. A human expert, knowledgeable about the semantics of concepts in both ontologies, validates the suggested matches generated by ArtGen using a GUI tool. The expert can either accept the match, keep the match but modify the suggested relationship between the matched terms, delete a suggested match or say that the match is irrelevant for the application at hand. The expert can also indicate new matches that the articulation generator might have missed.

The process of constructing an articulation is an iterative process and after the expert is satisfied with the rules generated, they are stored and used when information needs to be composed from the two ontologies. The response of the expert is also logged and the articulation generator uses the expert's feedback to generate better articulations in future while articulating similar ontologies for similar applications. This learning process improves the quality of future generation of articulations from similar information sources.

The heuristic matchers used by the automated articulation generator can be classified into two broad types - iterative and non-iterative. Since the articulation generator is modular in nature, any application-specific matching algorithm can be plugged in. However, we believe that a set of basic matching algorithms will be useful in a wide variety of applications and we experimented to determine such a set.

3.1 Non-iterative Algorithms

Non-iterative algorithms are ones that identify the matching concepts in the two ontologies in one pass. Our linguistic matcher employs only non-iterative algorithms.

3.1.1 Linguistic Matching

The linguistic matcher looks at all possible pairs of terms from the two ontologies it is matching and assigns a similarity score to each pair. If the similarity score is above a threshold, then the match is accepted and an articulation rule is generated. The threshold can be modified by the expert performing the articulation to increase or decrease the number of matches generated.

We expect that a concept name is represented as a string of words. The matcher constructs all possible pairs of words where the two words in a pair come from different strings. The matcher uses a word-similarity table generated by a *word relator* which we describe below. It looks up a word-similarity table to determine the similarity between all such pairs of words. Finally, it computes the similarity of the strings based on the similarity of the pairs of words.

- match(String s1, String s2, WordSimilarityTable wst)
  - List similarityList;
  - for each word w1 in s1:
    - for each word w2 in s2:
      similarityScore ← wst.lookup(w1, w2);
      Add (w1, w2, similarityScore) to similarityList;
  - Sort similarityList on the similarity score of the tuples;
  - Set matchedWords ← null;
  - floatingPointNumber matchingScore ← 0.0;
  - for each tuple (w1, w2, ss) in similarityList:
    - if either w1 or w2 is in matchedWords continue;
    - else
      matchingScore ← matchingScore + ss;
      add w1, and w2 to matchedWords;
  - similarityScore ← similarityScore / min(size(s1), size(s2));
  - return similarityScore;

For example, given the strings "Department of Defence" and "Defense Ministry", we see that $match(Defence, Defense) = 1.0$. Similarly, we have $match(Department, Ministry) = 0.4$. Therefore, we calculate the similarity between the two strings as:

```
match("Department of Defence",
      "Defense Ministry") = (1 + 0.4)/2 = 0.7.
```

The denominator is the number of words in the string with less number of words.

This similarity score of two strings is then normalized with respect to the highest generated score in the application. The normalization step removes the bias of word-relators that give very low similarity scores for all pairs of words or those that give very high scores to all pairs of words. If the generated similarity score is above the threshold, then the two concepts are said to match, and we generate an articulation rule, $(Match"DepartmentofDefence""DefenseMinistry"), 0.7$, the last number gives the confidence measure with which the articulation generator generated this match. The confidence measure varies betwen 0 and 1.

Constructing the Word-Similarity Table:
We have experimented with several ways to generate the table containing the similarity between all pairs of words. After checking if the words are spelt similarly, we derive word similarity using methods that can be differented into two main groups: a) thesaurus based, b) corpus-based.

Thesaurus-Based Word-Relator: We have devised matching algorithms based on dictionaries or semantic networks, like Nexus [6] and WordNet [1]. WordNet gives us a list of synonyms for each word. If the two words are found to be synonyms, then we return a similarity score of 1.0. If the two words are not synonyms, we look at the the number of words that are "similar" in the defnitions of each word. This process of looking into the definitions of words to find their similarity can be repeated recursively until a fixed-point is reached or uptil a specified depth is reached at which point we require "similar" to be "same".

- GenerateSimilarity(word w1,word w2,dictionary dict,depth dep)
  - if (w1 == w2) return 1;
  - if (dep == 0) return 0;
  - else
    - def1 ← dict.lookup(w1);
    - def2 ← dict.lookup(w2);
    - List similarityList ← new List;
    - for each word wd1 in def1:
      - for each word wd2 in def2:
        Add (w1, w2, GenerateSimilarity(wd1, wd2, dep-1)) to similarityList;
    - Sort similarityList on the similarity score of the tuples;
    - Set matchedWords ← null;
    - floatingPointNumber matchingScore ← 0.0;
    - for each tuple (w1, w2, ss) in similarityList:
      - if either w1 or w2 is in matchedWords continue;
      - else
        matchingScore ← matchingScore + ss;
        add w1, and w2 to matchedWords;
    - similarityScore ← similarityScore / min(size(def1), size(def2));
    - return similarityScore;

For example, the definitions of "truck" and "boat" are "an automotive vehicle suitable for hauling",and "a vessel for water transportation". If the specified depth is 1, we do not look into the definitions of "vehicle" and "vessel" to determine their similarity. Since they are not exactly the same, we say their similarity is 0. If however, the depth were set to 2 (or more), we would look up the definitions of "vehicle" and "vessel", discover their definitions both have "transportation" in common, and generate a similarity measure and propagate that similarity up to generate a non-zero similarity for "truck" and "boat".

Corpus-Based Word Relator: Word similarities used by the linguistic matcher can also be generated using a corpus-based matching algorithm. The word relator uses a corpus of documents belonging to the domain of the ontologies that are being matched. The terms that appear in the ontology should also appear in the documents. The word relator calculates word-similarity scores based on the similarity of the contexts in which the words appear in the documents [13].

We identify the context in which a word, w, appears by looking at words that appear in a 1000-character neighbourhood of all occurrences of w in documents in the corpus. For example, the words "in", "the", "For", and "example" constitute the 30-character neighbourhood of the word "corpus" at the end of the last sentence. In the example, we looked at a 15-character window ahead of the word and 15 characters behind the word and chose all words that are complete in these windows. Therefore, even though part of the word "documents" appears in the 15-character window before the word "corpus" in that sentence it is ignored.

We look at all words that appear in the corpus. For each occurrence of a word, we identify the words in its context. The number of rows in the context vector, V_w, of a word w is equal the number of words in the corpus. Let $V_w[i] = c$. This implies that the i^{th} word in the corpus occurs with a frequency c in the 1000-character neighbourhood of the word w. The cosine of such normalized context vectors of two words gives a measure of the similarity of contexts in which the two words appear. We use this similarity measure to generate a table of word similarities that is then used by the linguistic matcher.

Ideally, we would have one corpus associated with one ontology, where the documents in the corpus use the terms in the exact sense as it is used in the ontology. However, for our experiments we did not have such a domain-specific corpus. We generated a corpus by searching the web(google) using 5 keywords each from the two ontologies that we were seeking to articulate. Typically, a corpus of 200 pages proved adequate to produce good matches.

3.1.2 Instance-based Heuristics

Instance-based matching heuristics have been used to successfuly match schemas in databases [14]. Such matchers look at data types, and extract other features like lengths of attributes, numerical or lexical statistics of attributes, and match classes based on such feature vectors. Though, we can handle ontologies, whose concepts also have instances associated with them, oftentimes, businesses are reluctant to make instances available. Thus, we have designed our algorithms assuming that no instance data is available. However, if such information is available, the matcher can be extended to use instance information.

3.2 Iterative Algorithms

Iterative algorithms are algorithms that depend upon existing articulation rules to generate further articulation rules. They require multiple iterations over the two source ontologies in order to generate semantic matches between them.

3.2.1 Structure-based Heuristics

These algorithms look for structural isomorphism between subgraphs of the ontologies to find matching concepts. For the ontologies we have experimented with, we see that a purely structural matcher - one that simply looks for isomorphism between subgraphs in the ontologies without considering concept names- performs very poorly and is inadequate.

Therefore, we propose a structure-based matcher that is called after the matches generated by a linguistic matcher is available. If the linguistic matcher has matched nodes, "A" and "B" in the ontology-graphs, the structural matcher looks to match their children (also parents), "C", and "D", if they have not already been matched. If a substantial percentage (above the threshold supplied) of the parents of "C" have been matched with those of "D", and the children of "C" have been matched with those of "D", then an articulation rule matching "C" and "D" is generated.

3.2.2 Inference-based Heuristics

An inference engine can reason with the rules available with the ontologies and any seed rules provided by an expert ontologies to generate matches between the ontologies. For example, a rule:

```
(=> (InstanceOf X O1.LuxuryCar)
    ((InstanceOf X O2.Car) AND
     (O2.PriceOf Y X) AND
     (O2.UnitOf X "$") AND
     (ValueOf X Z) AND
     (> Z 40,000)))
```

which says that any instance of $O1.LuxuryCar$ is an instance of $O2.Car$, that has a price greater than $\$40,000$.

4 Experiments & Results

We have implemented the linguistic methods and the structural methods in our articulation generator (using Java as the programming language). We experimented with three sets of ontologies represented in RDF[2]:

1. Ontologies (avg. 30 nodes) constructed manually to represent a domestic airlines (terminology used on United Airlines website) and a airforce ontology (terminology used in the US Air Force).
2. Ontologies (avg. 50 nodes) constructed manually from the NATO government web-sites representing each web-page associated with an department of the government as a node. The edges in the ontology graph were derived from the links between the pages.

We measured the accuracy of the generated match by comparing the results generated by the automated matcher with those expected by the expert. Any match deleted by the expert was taken to be a false positive and lowered the precision figures, and a match added by the expert that the automated generator failed to find lowered the recall. We summarize the results of the several experiments below:

- A purely structural method which requires exact concept-name match, like that has been used in existing tools, fails to generate even 50% of the matches expected by the expert. This result is not surprising since despite having useful information, the structure of the ontologies used hardly encode sufficient semantics to use them solely for ontology alignment.
- Adding linguistic heuristics gave significantly better results, especially, the corpus-based heuristic provided we supplied the matcher with a good representative set corpus of documents from the applicable domain.
- However, a multi-strategy approach works best. On the average about 75% of the matches were generated, with less than 5% false positives that the expert indicated was not correct. The linguistic method generates on the average about 60-70% of the matches (recall with 95% precision). Adding the structural matcher, boosts the matches by 5-10%. The human expert provided us with the other 30% of the rules that were not generated automatically.

The performance of the algorithm depends upon several parameters:

- Thesaurus-based Method: A general purpose thesaurus results in very poor results. Domain-specific thesauri produce better results but might not be available.
- Corpus-based Method: A corpus-based method produced better results than the thesaurus-based method. In the aircraft example, solely employing the thesaurus-based method produced a 30% recall (at 90% precision). A corpus-based method, where we obtained a corpus by searching the web with a few key-words from the domains, boosted the match to 60%. Combining the two, we obtained a recall of 70%.
- Scalability: Initially, we tried the corpus-based method with a preprocessing step of collecting the corpus and building up the word-context vectors. The linguistic matcher, while matching the ontologies, constructed the word similarities as needed. However, for a test case with 300 nodes in each ontology took an hour to run on a Pentium III machine with 256M memory. It becomes clear that for larger ontologies, the algorithm does not scale well if we compute the word similarities while matching the ontologies. For the algorithm to scale, not only, do we need to build the corpus and construct the word-context vectors a priori, but also pre-compute the similarity of all pairs of words in the corpus. The corpus-based method can then be thought of as equivalent to a lookup based method, where the word-similarity matrix is constructed from the words in the corpus. This variation of the corpus-based method scaled well and for our ontologies finished within a couple of minutes at worst.
- Quality: The quality of the matches were very dependent on the quality of the corpus available. We experimented with corpuses of size 50 pages, 100 pages, 200 pages and 1000 pages. Corpuses of size 50-100 pages resulted in low recall figures for the matches. A size of 200 webpages often proved adequate to generate a recall of 70%, although in most cases having a corpus of 1000 matches increased the recall, it was less than a few percentages.

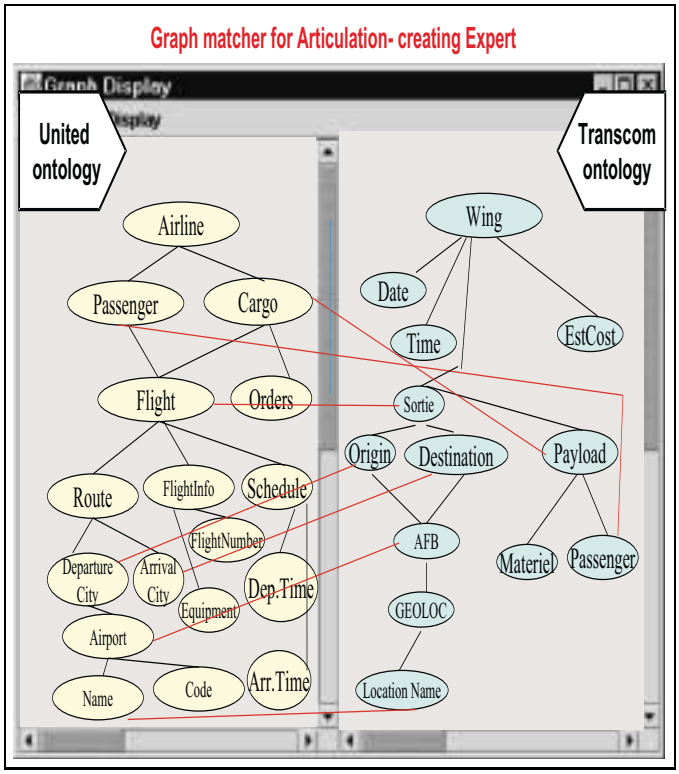

Figure 2. Example of an articulation of United Airlines Ontology with TRANSCOM Ontology

In Figure 2(hand-drawn), we show two ontologies - the United Ontology and the TRANSCOM Ontology and the matches generated. We used a hybrid method that uses WordNet as a thesaurus, and a corpus generated by searching google. For example, the page "http://www.etrackcargo.com/Help/Agents/Fieldwas part of the corpus. The confidence scores of the matches are as follows when the threshold was set to 0.7:

If the threshold was set to a lower value 0.60, we introduced false positives like $(Match\ Airline\ Destination\ 0.61)$. Further lowering the threshold to 0.50 introduces more false matches $(Match\ FlightNumber\ Sortie\ 0.52)$, $(Match\ Equipment\ Materiel\ 0.54)$. Only the first two matches were generated using a word-relator that consults WordNet. We ran the word-relator with a depth value of 1. That is, the relator looks into the definition of the two words for similar words but does not proceed any further recursively. The match between $Cargo$

Table 1. The matches between the United and TRANSCOM Ontologies

Term in United.ont	Term in TRANSCOM.ont	Confidence Score
Passenger	Passenger	1.0
Cargo	Payload	1.0
Departure Time	Time	0.90
Arrival Time	Time	0.88
Arrival City	Destination	0.79
Name	Location Name	0.75
Departure City	Origin	0.72
Airport	Airforce Base	0.71
Flight	Sortie	0.70

and $Payload$ was not higher than 0.7 using the corpus-based word-relator and would not have been suggested. Thus, we see that a hybrid method gives us a better accuracy than any one method alone.

In this example, we see that with a threshold value of 0.7, we generate all the desired matches and no false matches - the ideal solution. However, acheiving a 100all cases. From this and several other experiments, we see that setting a threshold of 0.7 gives the most number of matches with the a 95the matches are false positives. However, in a significant number of cases the value of the threshold varies depending upon both the corpus supplied and the ontologies being matched. Therefore, we suggest that for an unknown application or an unknown corpus, when running the first time, the matching threshold be set to 0.7. This is not to say that a threshold of 0.7 will produce best results always but from our experience it provides a good starting point as there is no one threshold value that will provide satisfactory for all applications. If not satisfied with the results the expert can then increase or decrease the threshold to get better matches.

5 Conclusion

We discussed several heuristic methods to produce simple matching rules between concepts in ontologies that are being aligned. We see that a multi-strategy method based on intial linguistic-similarity followed by structural matching generates matches between ontologies with reliable accuracy. The work of an expert who then validates the suggested rules or supplies new rules is substantially reduced by the automated component.

REFERENCES

[1] 'Wordnet - a lexical database for english. http://www.cogsci.princeton.edu/ wn/', Technical report, Princeton University.

[2] 'Resource description framework(rdf) model and syntax specification, w3c recommendation http://www.w3.org/tr/rec-rdf-syntax', (1999).

[3] M. D. Siegel C. H. Goh, S. E. Madnick. Semantic interoperability through context interchange: Representing and reasoning about data conflicts in heterogeneous and autonomous systems http://citeseer.nj.nec.com/191060.html.

[4] H. Chalupsky, 'Ontomorph: A translation system for symbolic knowledge', in *KR 2000*, pp. 471–482. Morgan Kaufmann Publishers, (Apr 2000).

[5] A. Doan, P. Domingos, and A. Y. Halevy, 'Reconciling schemas of disparate data sources: A machine-learning approach', in *SIGMOD 2002*, (2001).

[6] J. Jannink, *A Word Nexus for Systematic Interoperation of Semantically Heterogeneous Data Sources*, Ph.D. dissertation, Stanford University, 2000.

[7] J. Madhavan, P. A. Bernstein, and E. Rahm, 'Generic schema matching with cupid', in *VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases, September 11-14, 2001, Roma, Italy*, pp. 49–58. Morgan Kaufmann, (2001).

[8] D.L. McGuiness, R.Fikes, J. Rice, and S. Wilder., 'The chimaera ontology environment', in *Seventh National Conference on Artificial Intelligence (AAAI-2000)*, (2000).

[9] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm, in *Proceedings of the Twelfth International Conference on Data Engineering, San Jose, CA*. IEEE Computer Society, (February 2002).

[10] N.F. Noy and M.A. Musen, 'Prompt: Algorithm and tool for automated ontology mergin and alignment', in *Seventh National Conference on Artificial Intelligence (AAAI-2000)*, (2000).

[11] D.E. Oliver, Y. Shahar, E.H. Shortliffe, and M.A. Musen, 'Representation of change on controlled medical terminologies', in *Proc. AMIA Conference*, (Oct. 1998).

[12] Yannis Papakonstantinou, Hector Garcia-Molina, and Jeffrey D. Ullman, 'Medmaker: A mediation system based on declarative specifications', in *Proceedings of the Twelfth International Conference on Data Engineering, February 26 - March 1, 1996, New Orleans, Louisiana*, ed., Stanley Y. W. Su, pp. 132–141. IEEE Computer Society, (1996).

[13] Hinrich Schuetze, 'Dimensions of meaning', in *Supercomputing*, pp. 787–796, (1992).

[14] L. L. Yan, R. J. Miller, L. M. Haas, and R. Fagin, 'Data-driven understanding and refinement of schema mappings', in *ACM SIGMOD*, (2001).

Attribute meta-properties for knowledge sharing

Valentina Tamma and **Trevor J.M. Bench Capon** [1]

Abstract. Formal ontological analysis is a methodology that builds on some philosophical notions in order to guide the process of building ontologies whose structure is correct and little or no tangled.
This paper presents an ontology model that facilitates formal ontological analysis, by providing a set of *metaproperties* which characterise the behaviour of concept properties in a concept definition, while providing a richer semantics of the concept. We describe concepts in terms of their attributes (characterising features) and we also describe the role played by these features in the concept definition, whether they are prototypical or exceptional, whether they are permitted to change over time, and if so, how often this happens, how likely is a concept to show these features, etc. We show that these metaproperties can support a methodology, OntoClean [44] that uses formal ontological analysis to build cleaner taxonomies (which are thus more sharable). The set of metaproperties for attributes we propose can be used to guide in determining which metaproperties for concepts hold for an ontology and therefore can support the use OntoClean.

1 Introduction

Many current applications such as e-commerce or the semantic web rely on the ability of different resources or agents to interoperate with each others and with users. In some cases, interoperation becomes more complex, because agents may have been independently developed, therefore the assumption that agents use the same communication language and the same terminology in a consistent way cannot be made. When dealing with independently developed agents, their interoperability with humans and others depends on the agents' ability to understand them, which leads us directly to ontologies. Ontologies are an explicit, formal specification of a shared conceptualisation, where a 'conceptualisation' refers to an abstract model of some phenomenon in the world by having identified the relevant concepts of that phenomenon, 'explicit' means that the type of concepts used, and the constraints on their use are explicitly defined, 'formal' refers to the fact that the ontology should be machine-readable, and lastly 'shared' reflects the notion that an ontology captures consensual knowledge, that is it is not private to some individual, but accepted by a group [37]. That is ontologies provide a formally defined specification of the meaning of those terms that are used by agents during the interoperation.
Agents can differ in their understanding of the world surrounding them, in their goals, and their capabilities, but they can still interoperate in order to perform a task. The interoperation among agents is the result of reaching an agreement on a shared understanding, mainly obtained by the reconciliation of the differences. This kind of reconciliation might be accomplished by *merging* the ontologies to which the agents involved in the interoperation refer to, that is, by building a single ontology that is the merged version of different agent's ontologies, which often cover similar or overlapping domains [8].
Ontology merging starts with the attempt to find the places in which the source ontologies overlap [24], that is the coalescence of two semantically identical terms in different ontologies so that they can be referred to by the same name in the resulting ontology. This is the only step of the merge process which is relevant to the scope of this article. The coalescence of terms in diverse ontologies has to be accomplished bearing in mind that agent's ontologies might be heterogeneous, and any kind of heterogeneity has to be reconciled in order to share knowledge. Heterogeneity is out of the scope of this article, however we recognise that it can hinder attempts to coalesce terms, especially when it concerns semantics. Ontology or semantic heterogeneity occurs when different ontological assumptions about overlapping domains are made [43].
Any consideration on ontology heterogeneity it is usually done assuming that the ontologies involved in the merging process are either built according to some kind of engineering methodology, such as Methontology [6], or ontology taxonomic structures are validated according to some methodologies such as OntoClean [44]. Both methodologies are aimed to insure that the ontology obtained after applying them is correct, that it does not contain cycles or recursive definitions, and it has a taxonomic structure that is no or little tangled.
Methontology and OntoClean are complementary methodologies in that Methontology provides the guidelines for building or re engineering ontologies, whereas OntoClean can be used either in the validation step (when ontologies are engineered or restructured) or simultaneously with the ontology construction (when ontologies are built from scratch). These two methodologies are currently undergoing an integration process [5] as part of the activities of the OntoWeb special interest group on Enterprise-standards Ontology Environments (SIG's home page: `http://delicias.dia.fi.upm.es/ontoweb/sig-tools/index.html`).
Methodologies to obtain well-built ontologies, however, are not enough to support the semi-automatic coalescence process. In fact in order to recognise whether two concepts (that can be affected by heterogeneity) are similar, we cannot only rely on the the terms denoting them, on the relationships with other terms, and on their descriptions, but we need to have a full understanding of the concepts. As noted by McGuinness [23], an explicit representation of the semantics of terms would be useful to understand whether two concepts are similar. It emerges that the current ontology models are not expressive enough to provide such an explicit representation of the semantics. Even when heavyweight ontologies are considered (that is, concepts described in terms of attributes, linked by relations,

[1] Department of Computer Science, University of Liverpool, Chadwick Building, Liverpool L69 7ZF, UK, email: {valli, tbc}@csc.liv.ac.uk

organised into an Is-a relationship and constrained by axioms) their expressiveness does not allow a full account of the semantics of the concepts described.
This paper is organised as follows: Section 2 presents the OntoClean methodology and the notions of formal ontological analysis, while Section 3 introduces our proposal for an ontology model encompassing a set of metaproperties for attributes which are discussed in the following subsections. This ontology model was also presented in [39], in this paper we do not discuss any implementation issues and we concentrate on the metaproperties, clarifying the relationship with the concept metaproperties used in OntoClean and the role attribute's metaproperties play in associating senses to concepts. Section 4 discusses the metaproperties and relates them with two notions (identity and rigidity) of formal ontological analysis and with roles. Then we proceed by presenting in Section 5 and subsections a novel approach to knowledge sharing that we are currently investigating and which motivated the ontology model presented in Section 3. This approach, called *ontology clustering*, is thought of being more suited to open evironments in which agents interoperate with each others. We Finally, Section 6 draws conclusions and in Section 7 we describe future work.

2 The philosophical notions of Identity, Unity, Essence, and Dependence

OntoClean [44] is a methodology to perform a *formal ontological analysis* on taxonomies in order to to verify which formal metaproperties hold, thus making clear and explicit the modelling assumptions made while designing the ontologies. The clarification and explication of the modelling assumptions is a necessary step to perform in order to evaluate ontologies, it permits knowledge engineers to detect and reconcile ontological conflicts that may affect one or more ontologies. Ontological conflicts may become apparent when two ontologies are compared in order to coalesce term, and they reveal cases of ontological heterogeneity. For example two well known ontologies, present the following conflict: one models Physical Object as subconcept of Amount of matter wheres the other models Amount of matter as subconcept of Physical object, this is a case of ontology heterogeneity due to different modellings of the concepts. Ontologial conflicts need to be detected and resolved if terms are to be coalesced.
OntoClean is strongly based on the philosophical notions of *identity, unity, essence (rigidity)*, and *dependence*. The attribute metaproperties we present in this paper are related to these notions, and we discuss them below.

Identity: Identity is the logical relation of numerical sameness, in which a thing stands only to itself. Based on the idea that everything is what it is and not anything else, philosophy has tried for a long time to identify the criteria which allow a thing to be identified for what it is even when it is cognised in two different forms, by two different descriptions and/or at two different times [45, 15]. This comprises both aspects of finding constitutive criteria (which features a thing must have in order to be what it is), and of finding re-identification criteria (which feature a thing has to have in order to be recognised as such by a cognitive agent). These are distinct, although equally important aspects of identity.In fact, while identity is not affected by the context and is based on the the intrinsic features of an object, whereas re-identification is affected by context and it is based on features that are external to the object. For example, an identity criterion for people is to have matching fingerprints, so two people are the same if they have the same fingerprints. Fingerprints are intrinsic to the individual, they are not assigned by an external agent. A re-identification criterion might depend on the role played by the object: one can be a student and an employee at the same time, and is re-identified as student by the student id, whereas is re-identified as employee by an employee number.
Although the problem of *identifying* what features an entity should have in order to be what it is and recognised as such has been central to philosophy, it did not have the same impact in conceptual modelling and more generally AI. The ability to identify individuals is central to the modelling process, more precisely, it is not the mere problem of identifying an entity in the world that is central to the ontological representation of the world, but the ability to *re-identify an entity in all its possible forms*, or more formally *re-identification in all possible worlds*. [2] That is, the problem is related to distinguishing a specific instance of a concept from its siblings on the basis of certain *characteristic properties* which are unique and intrinsic to *that instance* in its whole. Intrinsic properties correspond to the modelling primitive *attributes*. Extrinsic properties represent relations between classes, thus corresponding to the modelling primitive *relationship*.
This notion is, of course inherently time dependent, since time gives rise to a particular system of possible worlds where it is highly likely that the same instance of a concept exhibits different features [3]. This problem is known as *identity through change*: an instance of a concept may remain the same while exhibiting different properties at different instants of time. Therefore it becomes important to understand which features or properties can change and which cannot [44], and also the situations that can trigger such changes. If we reformulate the identity problem as *re-identification* we realise that re-identification is also affected by time; how can we re-identify the same instance at different instant of times? We face the re-identification problem in everyday life; we are able to recognise the features that permits us to distinguish an instance from the others, and when intrinsic features are not available, we 'attach' artificial features, that permit us to establish identity. One example is the *Student ID*, which is assigned to university students, in order to identify students univocally.

Unity: the notion of *unity* is often included in a more generalised notion of identity, although these two notions are different. While identity aims to characterise what is unique for an entity of the world when considered as a whole, the goal of unity is that of *distinguishing the* parts *of an instance from the rest of the world by means of a* unifying relation *that binds them together (not involving anything else)* [44]. For example, the question 'Is this my car?' represents a problem of identity, whereas the question 'Is the steering wheel part of my car?' is a problem of unity. Also the notion of unity is affected by the notion of time; for example, can the parts of an instance be different at different instants of time?

Essence: The notion of *essence* is strictly related to the notion of *necessity* [16]. An *essential property* is a property that is necessary for an object, that is, a property that is true in every possible world [22]. Based on the notion of *essence*, Guarino and colleagues [14] have introduced the notion of *rigidity*. A rigid property is a

[2] Some philosophers, e.g. Lewis [21, page 39 ff], hold that there is no such thing as trans-world identity, although objects in one world can have *counterparts* in other worlds.
[3] Here the counterpart theory does not hold, and so identity through time is always accepted.

property that is necessary to all instances in any instant of time, that is a property ϕ such that: $\Box(\forall x,\ t\phi(x,\ t) \rightarrow \Box\forall t'\ \phi(x,\ t'))$. For this formula, and in the remainder of this paper, we use the modal notions of *necessity* $\Box$ and *possibility* $\diamond$ quantified over possible worlds (in Kripke's semantics [18]), meaning that the extension of predicates concerns what exists in any possible world. We use these operators according to the following meanings: $\Box\ \phi$ means that ϕ holds in *all* possible worlds $\diamond\ \phi$ means that ϕ is possible, i.e. that ϕ holds in *at least* one possible world.
Rigidity strictly depends on the notions of *time* and *modality* [38]; this point is further elaborated in Section 4.2. It is important,however, not to confuse modal necessity with temporal permanence. Modal necessity means that the property is true in every possible world. Time is undoubtedly one partition of these worlds, but temporal permanence means that the property is true in that world (time), with no information concerning the other possible worlds, and this might happen by pure chance.

Dependence: In OntoClean [44], the notion of dependence is considered related to concept properties. In this context, dependence permits us to distinguish between *extrinsic* and *intrinsic* properties based on whether they depend on objects other than the one they are ascribed to or not.

In order to establish whether these metaproperties hold, OntoClean is supported by a description logic based system that can help knowledge engineers to assign the metaproperties to concepts and to verify the taxonomic structure on the grounds of the modelling methodology. In this paper we focus our attention on the process of assigning the metaproperties. OntoClean guides knowledge engineers in this process by asking them to answer some questions such as "Does the property carry identity". Knowledge engineers can answer yes, no or unsure, in this latter case more specific questions can be asked, such as "Are instances of the property countable?".
The OntoClean methodology depends on the knowledge engineers understanding of the ontologies to analyse and can thus be problematic if used to evaluate independently designed ontologies. Moreover, OntoClean does not take into account the structure of concept definitions, as it does not consider the characteristic features (or *attributes*) that might have been used to define concepts.
This work proposes an enriched ontology model whose aim is to complement the OntoClean methodology, by providing an additional way to determine metaproperties to concepts. In our proposal we describe concepts in terms of their characterising properties, which are in turn described not only in terms of their structural features (such as range, domain, cardinality etc.), but also in terms of their metaproperties, which describe the contribution given by these properties to the concept definition. We describe the enriched ontology model and the metaproperties for attributes in the next sections.

3 Enriched ontology model

The ontology model we propose comprises *concepts*, *attributes*, *relations*, and *instances*. We do not consider here axioms. Concepts represent the entities of the domain and the tasks we want to model in the ontology. Concepts are described in terms of defining properties, which are represented by associating an *attribute* with either a single value or a set of values. Concepts are organised into an Is-a hierarchy, so that a concept attributes and their values are inherited by subconcepts. Multiple inheritance is permitted, so attributes and their values can be inherited from multiple parents. The values associated with an attribute can be restricted in order to provide a better definition of a concept [19].
Attributes are described in terms of their structural characteristics, such as the concepts that they are defining, their allowed values, the type of the values (string, integer, etc.), and the maximum and minimum values (if attributes are numeric). Attributes are also described in term of the following metaproperties:

- *Attribute's behaviour over time*: The metaproperties *Mutability, Mutability Frequency, Event Mutability* and *Reversible Mutability* provide a better description of attributes by characterising their behaviour over time, that is, whether they are allowed to change their value during the concept lifetime (*Mutability*) and how often the change occurs *Mutability Frequency*), whether the change is reversible (*Reversible Mutability*), and what triggers change (*Event Mutability*);
- *Modality*: this meta-property is a qualitative description of the degree of inheritability of a concept property by its subconcepts;
- *Prototypes* and *Exceptions*: the metaproperties *Prototypical* and *Exceptional* aim to describe properties that are prototypical for a concept, that is the properties that obtain for the *prototypical* (from a cognitive viewpoint, according to Rosch [30]) instances of a concept. Exceptions are those properties which can be ascribed to a concept although being highly unusual;
- *Inheritance* and *Distinction*: *inherited* metaproperties regard those properties that hold because inherited from an ancestor concept, they may be overruled in the more specific concept in order to accommodate inheritance with exceptions. *Distinguishing* are those properties that permit us to distinguish among siblings of a same concept. In other words a distinguishing property ϕ is a property such that $\diamond\exists x\ \phi(x)\ \wedge\ \diamond\exists x\ \neg\phi(x)$, that is there is possibly something for which the property ϕ holds, and there is possibly something for which the property does not hold, and these are neither tautological nor vacuous [44]. Distinguishing properties might cause disjoint concepts in the ontology's taxonomic structure.

These metaproperties provide means to distinguish between *necessary* and *sufficient* conditions for class membership. Indeed, the modality meta-property and those describing the behaviour over time permit the identification of essential (or rigid) properties and necessary properties are those that are essential to all instances of a concept. Prototypical properties are good candidates to identify sufficient conditions, as discussed in Section 3.3.
Relations between concepts are supported by the model as are instances. Finally, the ontology model supports roles. Concepts are also used to represent *roles*, which can be thought of describing the *part played* by a concept in a context, (a more complete discussion on roles is postponed to Section 4.3). Roles are described in terms of their context, and the formal role relationship holds, that is, roles are related to concepts by a 'Role-of' relations.
This ontology model enriches the traditional model proposed initially by Gruber [12], in that it permits the characterisation of a concept properties. From this viewpoint it should be more expressive. The solution of adding information characterising concept properties is a controversial one. Although we do realise that often it is not true that 'more is better', this work claims that an ontology model which include this type of property's characterisation might be helpful to deal with ontology heterogeneity problems in two ways. On the one hand the model complements the set of formal ontological properties proposed in [44], and can guide in assigning these to concepts

in a way which depends on concept definitions in terms of attributes. This might result particularly useful when knowledge engineers need to assign formal properties to ontologies they have not designed.
On the other hand, this conceptual model for ontologies facilitates a better understanding of the concept semantics. Currently ontology merge is performed by hand based on the expertise of the knowledge engineers and on the ontology documentation. Even in this case the ontology model we propose can prove useful by providing a characterisation of the properties, which can help to identify semantically related terms. The following subsections describe all the metaproperties for attributes but Inheritance and Distinction (which are trivial) more in detail:

3.1 Behaviour over time

The metaproperties which model the behaviour of the attributes over time are:

- *Mutability*, which models the liability of a concept property to change, a property is mutable if it can change during the concept lifetime;
- *Mutability Frequency*, which models the frequency with which a property can change in a concept description;
- *Event Mutability*, which models the reasons why a property may change; *Reversible Mutability*, which models reversible changes of the property.

These metaproperties describe the behaviour of *fluents* over time, where the term *fluent* is borrowed from situation calculus to denote a property of the world that can change over time. Modelling the behaviour of fluents corresponds to modelling the changes in properties that are permitted in a concept description without changing the essence of the concept. Describing the behaviour over time also involves distinguishing properties whose change is *reversible* from those whose change is *irreversible*.
Property changes over time are caused either by the natural passage of time or are triggered by specific event occurrences. We need, therefore, to use a suitable temporal framework that permits us to reason with time and events. In [39] we chose *Event Calculus* [17] to accommodate the representation of changes. Event calculus deals with local event and time periods and provides the ability to reason about change in properties caused by a specific event and also the ability to reason with incomplete information.
Changes of properties can be modelled as *processes* [35]. Processes can be described in terms of their start and end points and the changes that happen in between. We can distinguish between *continuous* and *discrete changes*, the former describing incremental changes that take place continuously while the latter describe changes occurring in discrete steps called *events*. Analogously we can define *continuous properties* to be those changing regularly over time, such as the age of a person, versus *discrete properties* which are characterised by an event which causes the property to change. If a property's mutability frequency is *regular* (that is it changes regularly), then the process is continuous, if it is *volatile* the process is discrete, and if it changes *once only* in the concept lifetime, then the process is considered discrete and the triggering event is set equal to *time-point=T*.
Any regular occurrence over time can be, however, expressed in form of an event, since most of the forms of reasoning for continuous properties require discrete approximations. Therefore in the ontology model we present here, continuous properties are thought of as discrete properties where the event triggering the change in property is the passing of time from the instant t to the instant t'. Events are always thought of as *point events*, and we consider *durational events* (events which have a duration) as being a collection of *point events* in which the property whose mutability is modelled by the set of metaproperties hold as long as the event lasts.

3.2 Modality: Weighing the validity of attributes' properties

The term modality is used to express the way in which a statement is true or false, which is related to establish whether a statement constitutes a *necessary truth* and to distinguish necessity from possibility [18]. The term can be extended to qualitatively measure the way in which a statement is true by trying to estimate the number of possible worlds in which such a truth holds. This is the view we take in this work, by denoting the degree of confidence that we can associate with finding a certain world with the meta-property *modality*. This notion is analogous to the *rankings* defined by Goldszmidt and Pearl [10]: *'Each world is ranked by a non-negative integer κ representing the degree of surprise associated with finding such a world'*.
Here we use the term modality to denote the degree of surprise in finding a world where the property P holding for a concept C does not hold for one of its subconcepts C'. The additional semantics encompassed in this meta-property is important for reasoning with statements that have different degrees of credibility. Indeed there is a difference in asserting facts such as 'Cats are pets' and 'All felines are pets', the former is generally more believable than the latter, for which many more counterexamples can be found. The ability to distinguish facts whose truth holds with different degrees of strength is important in order to find which facts are true in every possible world and therefore constitute *necessary truth*.
The ability to evaluate the degree of confidence in a property describing a concept is also related to the problem of reasoning with ontologies obtained by merge. In such a case, mismatches can arise if a concept inherits conflicting properties. In order to be able to reason with these conflicts some assumptions have to be made, concerning on how likely it is that a certain property holds. In case of conflict the property's degree of credibility can be used to apply some forms of non monotonic reasoning or belief revision. For example, we could rank the possible alternatives on the grounds of the degree of credibility following an approach similar to the one presented in [10].

3.3 Prototypes, exceptions, and concepts

In order to get a full understanding of a concept it is not sufficient to list the set of properties generally recognised as describing a typical instance of the concept but we need to consider the known exceptions. In this way, we partially take the cognitive view of prototypes and graded structures, which is also reflected by the information modelled in the meta-property *modality*. In this view all cognitive categories show gradients of membership which describe how well a particular subclass fits people's idea or image of the category to which the subclass belong [30]. Prototypes are the subconcepts which best represent a category, while exceptions are those which are considered exceptional although still belonging to the category. In other words all the sufficient conditions for class membership hold for prototypes. For example, let us consider the biological category *mammal*: a *monotreme* (a mammal who does not give birth to live young) is an example of an exception with respect to the property of giving birth to live young. Prototypes depend on the context (that is on the specific domain that is conceptualised); there is no universal prototype but there are several prototypes depending on the context,

therefore a prototype for the category *mammal* could be *cat* if the context taken is that of *animals that can play the role of pets* but it is *lion* if the assumed context is *animals that can play the role of circus animals*. In the ontology model presented above the context can be partially described by the roles applicable to the concept for which prototypical and exceptional properties are modelled. By providing this example we do not mean that any member of the category *animals that can play the role of pets* could be a prototype, but just that prototypes vary if we vary the perspective we are taking on the domain. Therefore there is no unique prototype for the category *animal* but a number of prototypes, depending on how people conceptualise the domain, and this implies also contextual information, for example what is the role played by animals.
Ontologies typically presuppose context and this feature is a major source of difficulty when merging them, since information about context is not always made explicit.
Prototypes are also quite important in that they provide a frame of reference for linguistic quantifiers such as *tall*, *short*, *old*, etc. These quantifiers are usually defined or at least related to the prototypical instance of the concept which is being described, and indeed their definition changes if we change the point of reference.
Therefore including the notions of prototypes and exceptions permits us to provide a frame of reference for defining these qualifiers with respect to *a specific concept*. For the purpose of building ontologies, distinguishing the prototypical properties from those describing exceptions increases the expressive power of the description. Such distinctions do not aim at establishing default values but rather to guarantee the ability to reason with incomplete or conflicting concept descriptions.
The ability to distinguish between prototypes and exceptions helps to determine which properties are necessary and sufficient conditions for concept membership. In fact a property which is prototypical and that is also inherited by all the subconcepts becomes a natural candidate for a necessary condition. Prototypes, therefore, permit the identification of the subconcepts that best fit the cognitive category represented by the concept *in the specific context given by the ontology*. On the other hand, by describing which properties are exceptional, we provide a better description of the membership criteria in that it permits us to determine what are the properties that, although rarely holding for that concept, are still possible properties describing the cognitive category.
Prototypes and exceptions can prove useful in dealing with conflicts arising from ontology merging. When no specific information is made available about a concept and it inherits conflicting properties, then we can assume that the prototypical properties hold for it.

4 Discussion

The ontology model presented in previous section could be implemented in any kind of ontology representation formalisms. In [39] we presented an implementation of the ontology model above in a frame-based representation formalism, therefore attributes were described by associating values to slots, and their structural description and metaproperties were modelled by the slot's facets.
By adding the metaproperties to the ontology model, we provide an explicit representation of the attributes' behaviour over time, their prototypicality and exceptionality, and their degree of applicability to subconcepts. This explicit representation may be used to support and complement the OntoClean methodology [44], in that they can help in determining which metaproperties hold for concepts, as we will illustrate in remainder of this section.
Furthermore, the enriched ontology model we propose forces knowledge engineers to make ontological commitments explicit, that is the agreement on the meaning of the terms used to describe a domain [13]. Knowledge sharing is possible only if the ontological commitment of the different agents is made explicit. Real situations are information-rich events, whose context is so rich that, as it has been argued by Searle [32], it can never be fully specified. When dealing with real situations one makes many assumptions about meaning and context [31], and these are rarely formalised. But when dealing with ontologies these assumptions must be formalised since they are part of the ontological commitments that have to be made explicit. Enriching the semantics of the attribute descriptions with things such as the behaviour of attributes over time or how properties are shared by the subconcepts makes some important assumptions explicit.
The enriched semantics is essential to reconcile cases of ontology heterogeneity. By adding information on the attributes we are also aiming to measure the similarity between concepts more precisely and to disambiguate between concepts that *seem* similar while they are not.
A possible drawback of enriching the ontology model is that knowledge engineers are required a deeper analysis of a domain. We realise that it makes the process of building an ontology even more time consuming but we believe that a more precise ontological characterisation of the domain at least balances the increased complexity of the task. Indeed, in order to include the attribute's metaproperties to the ontology model, knowledge engineers need to have a full understanding not only of the concept they are describing, but also of the context in which the concept is used. Arguably, they need such knowledge if they are to perform the modelling task thoroughly.
The evaluation of the cost to pay for this enriched expressiveness and of the kind of reasoning inferences permitted by this model are strictly dependent on the domain and the task at hand. We can imagine that the automatic coalescence of terms might require more sophisticated inferences whose cost we cannot evaluate *a priori*. In some other cases, the simple matching between properties' characterisations might help in establishing or ruling out the possiblity of semantic relatedness. For example, two concepts are described by the same properties but with different characterisations, this might indicate that the concepts have been conceptualised differently.

4.1 Identity

The idea of modelling the permitted changes for a property is strictly related to the philosophical notion of *identity*. The metaproperties modelling the behaviour over time are, thus, relevant for establishing the *identity* of concept descriptions [44], since the proposed ontology model addresses the problem of modelling identity when time is involved, namely *identity through change*, which is based on the common sense notion that an individual may remain the same while showing different properties at different times [16]. The knowledge model we propose explicitly distinguishes the properties that can change from those which cannot, and describes the changes in properties that an individual can be subjected to, while still being recognised as an instance of a certain concept.
Prototypical and exceptional properties and the modality metaproperties describing how the property is inherited in the hierarchy can all contribute to determine what are the necessary and sufficient conditions for class membership. Necessary and sufficient conditions are ultimately the conditions that permit us to define the properties constitutive of identity and to distinguish them from those that permit re-identification.

In order to find suitable identity criteria (which permit to identify a concept), knowledge engineer should look at *essential property*, that is those properties which hold for an individual in every possible circumstance in which the individual exists. It is important to note that essential properties should also be intrinsic if they have to be used to determine identity.
Also inheritance and distinction contribute to identify identity conditions, in that identity conditions have to be looked for among distinguishing properties.

4.2 Rigidity

Identity through change is also relevant to determine *rigidity*. In Section 2 a *rigid property* is defined as *a property that is essential to* all *its instances*.
In [38] we have related the notion of *rigidity* to those of *time* and *modality*; and, by including in our ontology model a meta-property *modality* and that concerning the behaviour over time, we can precisely identify rigidity in the subset of the set of possible worlds. Indeed, since an ontology defines a vocabulary, we can restrict ourselves to the set of possible worlds which is defined as the set of maximal descriptions obtainable using the vocabulary defined by the ontology [26]. By characterising the rigidity of a property in this subset of possible worlds we aim to provide knowledge engineers the means to reach a better understanding of the *necessary* and *sufficient* conditions for the class membership. However, this does not mean that the rigidity of a property depends on any account of whether the property is used to determine class membership or not. That is, the final aim is to try to separate the properties constitutive of identity from those that permit re-identification. Under the assumption of restricting the discourse to this set of possible worlds, *rigid properties* are those properties which are inherited by all subconcepts, and thus which have a certain degree of belief associated with the meta-property *modality* and that cannot change in time.
It is important to note that, although in [39] we have modelled this information as a facet which can take value in the set {*All, Almost all, Most, Possible, A Few, Almost none, None*}, the choice of such a set is totally arbitrary, and it was meant to be such. Knowledge engineers should be able to associate with this meta-property either a probability value, if they know the probability with which the property is inherited by subconcepts, or a degree of belief (such as a κ-value, as in [10], which depends on a ϵ whose value can be changed according to the knowledge available, thus causing the κ function to change), if the probability function is not available.

4.3 Roles dependence on identity and rigidity

Rigidity is not only central in order to distinguish necessary truth but also to recognise *roles* from concepts. The notion of *role* is as central to any modelling activity as those of *objects* and *relations*.
A definition of role that makes use of the formal metaproperties and includes also the definition given by Sowa [34] is provided by Guarino and Welty. In [44] they define a role as: *' the properties expressing the* part played *by one entity in an event, often exemplifying a particular relationship between two or more entities. All roles are* anti-rigid *and* dependent*... A property ϕ is said to be anti-rigid if it is not essential to* all *its instances, i.e.* $\Box(\forall x,\ t\phi(x,\ t) \rightarrow \Diamond\exists\ t'\ \neg\phi(x,t'))$*...*
A property ϕ is (externally) dependent *on a property ψ if, for all its instances x, necessarily some instance of ψ must exist, which is not a part nor a constituent of x, i.e.* $\forall x\Box(\phi(x) \rightarrow \exists y\psi(y) \wedge \neg P(y,x) \wedge \neg C(y,x))$*'*, where $P(y,x)$ denotes that y is a *part* of x while $C(y,x)$ denotes that y is a *constituent* of x. In other words a concept is a role if its individuals stand in relation to other individuals, and they can enter or leave the extent of the concept without losing their identity. From this definition it emerges that the ability of recognising whether rigidity holds for some property ϕ is essential in order to distinguish whether ϕ is a role.
Roles may be 'naturally' determined when social context is taken into account, and the social context determines the way in which a role is acquired and relinquished. For example, the role of `President of the country` is relinquished differently depending on the context provided by the country. So, for example, in Italy the role may be acquired and relinquished only once in the lifetime of an individual, whereas if the country is the United Sates, the role can be acquired and relinquished twice, because a president can be re-elected. Social conventions may also determine that once a role is acquired it cannot be relinquished at all. For example, the role `Priest` in a catholic context is relinquished only with the death of the person playing the role. The ability to distinguish roles gives also a deeper understanding of the possible contexts in which a concept can be used. Recognising a role can be equivalent to defining a context, and the notion of context is the basis on which prototypes and exceptions are defined.
In [36] Steimann compares the different characteristics that have been associated in the literature with roles. From this comparison it emerges that the notion of role is inherently temporal, indeed roles are acquired and relinquished dependent on either time or a specific event. For example the object *person* acquires the role *teenager* if the person is between 13 and 19 years old, whereas a person becomes *student* when they enroll for a degree course. Moreover, from the list of features in [36] it derives that many of the characteristics of roles are time or event related, such as: an object may acquire and abandon roles dynamically, may play different roles simultaneously, or may play the same role several time, simultaneously, and the sequence in which roles may be acquired and relinquished can be subjected to restrictions. Indeed, what distinguishes a role from a concept, in the modelling process, is that a role holds during a specific span of time in which some property holds. For example, the role 'Student' is applicable only if the property of being registered to a university holds. Therefore, the metaproperties that model the behaviour over time permits the representation of the acquisition and relinquishment of a role.
For the aforementioned reasons, ways of representing roles must be supported by some kind of explicit representation of time and events. Indeed the proposed model provides a way to model roles as fluents; moreover, by modelling the reason for which a property change, we provide knowledge engineers the ability to model the events that constrain the acquisition or the relinquishment of a role.

5 A novel proposal to knowledge sharing

We have illustrated and discussed a ontology model which is enriched with metaproperties providing a better characterisation of attribute. This characterisation is meant to help in disambiguating heterogeneous concepts when merging ontologies, since we assume that *two concepts can be matched if*:

- their description comprises attributes with matching names (synonyms, the name of an attribute is included into the other, etc.);
- candidate matching attributes are described by matching structural definitions (range of the attribute, cardinality, etc.);

- candidate matching attributes show the same behaviour in modelling the concept, that is, the same metaproperties hold for the attributes.

Matching similar concepts plays a pivotal role in those approaches to knowledge sharing which rely on shared ontologies in order to perform the translation between concepts in heterogeneous ontologies. Usually, knowledge sharing is obtained by creating one shared ontologies to which all the agents commit. However, such an approach has been compared to imposing a standard and suffers from the same drawbacks [42]. In this paper we propose a novel architecture to knowledge sharing, which is thought to be more scalable and maintainable, and thus offers more support to the Semantic Web paradigm we have discussed in the Section 1.
In contrast to an approach in which all resources share one body of knowledge here we propose to locate shared knowledge in multiple but smaller shared ontologies. This approach is referred to as ontology-based resource clustering, or shortly, ontology clustering [33]. Resources no longer commit to one comprehensive ontology but they are clustered together on the basis of the similarities they show in the way they conceptualise the common domain. Thus, we have not one, but multiple shared ontologies aggregated into clusters.
Each cluster can be thought of as a micro-theory shared by all the agents that conform to that cluster. Each micro-theory is in turn generalised and they are all eventually generalised by the top-level ontology which is a standard upper ontology like the *Upper-Cyc* [20], so as to obtain a structure that is able to reconcile different types of heterogeneity. We discuss here the feasibility of building such a structure, and in particular, we have investigated the different similarity measures that can be used in order to build clusters of ontologies.
This approach is analogous to *modularisation* in software engineering and is thought of having the same advantages, which are:

- **Modularity/separability**: Each cluster is like a module in software engineering and represents a specific aspect of the domain;
- **Composability**: Different clusters are composed by generalising the concepts that are common to them. This is the first step to permit heterogeneous resources to communicate;
- **Scalability**: The addition of a new resource to the architecture requires only the production of the mapping rules between the ontology associated to the new resource and the cluster to which this resource belongs;
- **Impact of change minimisation**: If a concept description needs to be changed only the mapping rules between the updated ontology and the cluster to which this ontology belongs need to be rewritten;
- **Division of ontology authoring efforts**: Ontologies composing a cluster do not need to be authored by the same people as long as their concepts can be mapped into the concepts of the cluster.
- **Accommodation of diverse formalisations**: A cluster can be comprised of ontologies representing different formalisations of the same domain, such as different temporal ontologies.

This approach has not been tested yet, therefore we can only foresee some disadvantages:

- There is no methodology which permit to build the structure of ontology clusters;
- Complexity of the first order clustering problem from the machine learning viewpoint;
- Lack of semantic-sensitive similarity measure to use to assess the similarity among concepts;
- Lack of tools that can support the building of the ontology clusters.

5.1 Ontology clusters

Ontology clustering is based on the similarities between the concepts known to different resources, where each resource represents a different aspect of the domain knowledge. We assume that the ontologies modelling the resources are consistent, non-redundant, and well structured. We also assume that the ontologies have been built with a methodology including a formal evaluation step, such as Methontology [11]. We also assume that the ontologies are specified by using a language that conforms to the ontology model described above.
Since our resources need to communicate in a sensible fashion they are all supposed to be familiar with some high level concepts. We group these concepts in an ontology rooted at the top of the hierarchy of ontologies. As it describes concepts that are specific to the domain and tasks at hand we refer to this ontology as the application ontology (following Van Heijst and colleagues, [41]. These concepts are reusable within the application but not necessarily outside the application. The concept definitions in the application ontology are chosen from an existing top-level ontology, which in our case is WordNet [25]. The application ontology thus contains a relevant subset of WordNet concepts. For each concept one or more senses are selected, depending on the domain. If some resources share concepts that are not shared by other resources then this leads to the creation of two (or more) sibling ontologies. Each sibling is a consistent extension of its parent ontology, but heterogeneous with respect to its peers. We do not pose any restriction to the types of heterogeneity that can affect the ontologies.
A cluster is referred to as *a group of consistent ontologies (possibly one) in our structure and is described by an ontology which is shared by those composing the cluster*. Both ontology clusters and ontologies within each cluster are organised in a hierarchical fashion where each sibling cluster specialises the concepts that are in its parent cluster. However, while multiple inheritance is permitted within the ontologies, it is not permitted between ontologies, therefore the structure of clusters is a tree. In this structure, the lower level clusters have more precise concept definitions than the higher levels, making the latter more abstract.
Clusters are linked by *restriction or overriding* relations, that is concepts in one parent ontology are inherited by its children cluster, but overriding is permitted [42]. The link between the resources and the local ontologies, on the other hand, is different, and is a *mapping relation* as defined in [42], that is a function preserving the semantics.
Figure 1 illustrates an example of this structure, where Local Ont. are the local ontologies.
Since different siblings can extend their parent cluster concepts in different ways the cluster hierarchy permits the co-existence of heterogeneous (sibling) ontologies. Figure 1 illustrates this particular structure, where $Local\ Ont._1$, $Local\ Ont._2$, $Local\ Ont._3$, and $Local\ Ont._4$ are the local ontologies, $Shared_{12}$ is the ontology shared by the local ontologies 1 and 2. Analogously $Shared_{34}$ is the ontology shared by the local ontologies 3 and 4. $Shared_{1234}$ indicates the ontology shared by the two below that is $Shared_{12}$ and $Shared_{34}$, and in this example is the application ontology itself, here denoted by *Application Ontology*. If some ontologies share concepts that are not shared by other ontologies then there is a reason to create a new cluster. A new ontology cluster here is a child ontology that defines certain new concepts using the concepts

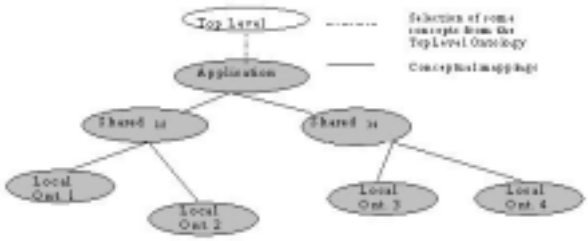

Figure 1. The hierarchy of multiple shared ontologies

already contained in its parent ontology. Ultimately, ontologies are likely to have concepts that are not shared with any other ontology. In our ontology structure, we then create a separate, domain-specific ontology as sub ontology of the cluster in which the ontology resides. We refer to these ontologies as local ontologies. The local ontologies are the leaf nodes of our ontology hierarchy. In each of the ontologies in the structure, concepts are described in terms of attributes and inheritance relations holding in the ontology's structure. Concepts are hierarchically organised and the inheritance (with exceptions) allows the passing down of information through the hierarchy. Multiple inheritance is only permitted within the ontologies.

Concepts are expressed in terms of *inherited* and *distinguishing* attributes. To the set of inherited attributes other attributes are added to distinguish the specific concept from the more general one. These attributes describe the characteristic differences between a concept and its siblings. The distinguishing attributes are used to map concepts from a source ontology into a target ontology preserving the meaning of the concept.

5.2 Towards the semi-automatic construction of ontology clusters

The structure of ontology clusters introduced in Section 5.1 builds on the ability of identifying similar concepts in different ontologies. Identifying which concepts are similar and assessing the degree of semantic similarity between them are, thus, two essential steps in the process of building ontology clusters. However, assessing the similarity between concepts in diverse ontologies is not a trivial task because of the heterogeneity that can affect concepts and their descriptions.

The problem of assessing semantic similarity has received much attention in the artificial intelligence field [27], [3]. In these efforts, 'semantic similarity' refers to a form of semantic relatedness using a network representation. In particular, Rada and colleagues [28] suggest that similarity in semantic networks can be assessed solely on the basis of the IS-A taxonomy, without considering other types of links. One of the easiest way to evaluate semantic similarity in taxonomies is to measure the distance between the nodes corresponding to the items being compared, that is the shorter the path between the nodes, the more similar they are. This way of assessing semantic similarity might be useful for semantic networks, however has the major drawback of computing the semantic distance between concepts which have a common ancestor, and thus it is not suitable for assessing the similarity of heterogeneous local ontologies that have to be clustered. Moreover, this method does not fully exploit the structure of the concept representation, since it does not take into account the concept description in terms of attributes, relationships, etc. thus making it more sensitive to synonym and homonym heterogeneity.

In fact, only few efforts are addressing the problem of facilitating the (semi) automatic reconciliation of different ontologies, and they have been mainly developed for merging different ontologies. Reconciling different ontologies involves finding all the concepts in the ontologies which are similar to one another, determine what the similarities are, and either change the source ontologies to remove the overlaps or record a mapping between the sources for future reference [9]. Similarity in these efforts is mainly lexical and not semantic. Most systems for ontology merging rely on dictionaries to determine synonyms, common substrings in the concept names, and concepts whose documentation share many unusual words. They do not take into account the internal structure of concept representation and the structure of the ontology.

The ontology merging environment Chimaera [24] partially considers the ontology structure in that it assess similarity between concepts also on the grounds of the subclass-superclass relationship and the attributes attached to the concept. Anchor-PROMPT [9] reconciles ontologies by finding *matching terms*, that is, terms from different source ontologies that represent similar concepts. Anchor-PROMPT assess both lexical and semantic matches exploiting the content and structure of the source ontologies (names of classes and slots, subclasses, superclasses domains and ranges of slot values, etc.), and the user's actions in merging the ontologies. However, the method used in Anchor-PROMPT is based on the assumption that if the ontologies to be merged cover the same domain, the terms with the same name are likely to represent the same concepts. Such an assumption is a good rule of thumb, but does not take into account cases of heterogeneity among the source ontologies. In fact, similar concepts might have different names, and be described by attributes with different names. Moreover, the hierarchical structure of the source ontologies might be different, thus a certain subclass-superclass relationship holding in one source ontology might not hold in the others.

The ontology model we have presented has been inspired by a particular approach to assess semantic similarity [29], where the authors propose a method for assessing semantic similarity which takes into account the differences in the level of explicitness and formalisation of the source ontologies specifications. This method does not require an *a priori* shared ontology, and thus makes it suitable for building the ontology clusters. The similarity between concepts in different sources ontologies is assessed by a matching process over synonym sets (thus accounting for lexical similarity), semantic neighborhood, and distinguishing features. The use of distinguishing features to assess similarity enables the authors not only to handle binary similarity measures, typical of lexical similarity (two terms are either similar or not), but also to consider gradients of similarity. This is based on the assumption that, in order for concepts to be considered similar, they should present some common features. By assessing similarity on the grounds of the distinguishing and common features, this method accounts for those problem of synonym terms heterogeneity that can affect both concepts and attributes.

In [29] the authors argue that from an analysis of different feature-based models for semantic similarity has emerged the necessity to account for the context dependence of the relative importance of distinguishing features and asymmetric characteristic of similarity assessments.

The method proposed by Rodríguez and Egenhofer is based on Tversky [40] matching process, which produces a similarity value that depends on both common and different characteristic. In order to take into account common and distinguishing features into the matching process, the usual ontology model is extended to include also an explicit specification of the features. By features the authors collectively mean the set of *functions*, *parts* and *attributes*. *Functions* represent the intended purpose of the instances of the concept they describe. For example the function of a university is to educate. *Parts* are the structural element of a concept, and they do not necessarily coincide with those expressing the *part-of* relationship, while *attributes* correspond to additional characteristics of a concept that are not considered to be neither parts nor functions.
It could be argued that enriching the concept structure by distinguishing between parts, functions and attributes can give rise to the articulation of new types of mismatches associated with the classifications of features. However, the authors claim that the advantages of enriching the concept structure, namely a matching process that compares corresponding characteristics of concepts, and the ability to distinguish different aspects of the context, modelled by the features, overweights the possible disadvantages deriving from a higher number of mismatches.
We believe that Rodríguez and Egenhofer approach to assess semantic similarity rises an important issue, which is that, in order to be able to have a better assessment of semantic similarity (that gives also gradients of similarity and not only a binary function) it is necessary to provide a richer description of the structure of the concepts in the source ontologies. However, we believe that the distinguishing features proposed in [29] overlap with the semantics already modelled by some relationships, such as *part-of*.

6 Conclusions

Sharing ontologies independently developed is a burning issue that needs to be solved. This paper presents a set of metaproperties describing concept characteristic features (attributes) that can be used to support both the process of building correct ontologies (by complementing and supporting the formal ontological analysis performed by the OntoClean methodology [44]) and the disambiguation of cases of ontology heterogeneity. Formal ontological analysis is usually demanding to perform and we believe that the set of metaproperties for attributes we propose can support knowledge engineers in determining the metaproperties holding for the concepts by forcing them to make the ontological commitments explicit.
The metaproperties we propose, namely Mutability, Mutability Frequency, Reversible Mutability, Event Mutability, Modality, Prototypicality, Exceptionality, Inheritance and Distinction encompass semantic information aiming to characterise the behaviour of properties in the concept description. We have argued that such a precise characterisation might help to disambiguate among concepts that only seem similar, and in turn can support mappings across the structure of multiple shared ontologies that we have devised as alternative to the current approaches to knowledge sharing. We claim that this characterisation of the concept properties is also very important in order to provide a precise specification of the semantics of the concepts. Such characterisation is essential if we want to perform a formal ontological analysis, in which knowledge engineers can precisely determine which formal tools they can use in order to build an ontology which has a taxonomy that is clean and not very tangled. The novelty of this characterisation is that it explicitly represents the behaviour of attributes over time by describing the permitted changes in a property that describe a concept. It also explicitly represents the class membership mechanism by associating with each attribute (represented in a slot) a qualitative quantifier representing how properties are inherited by subconcepts. Finally, the model does not only describe the prototypical properties holding for a concept but also the exceptional ones. By providing this explicit characterisation, we are asking knowledge engineers to make more hidden assumptions explicit, thus providing a better understanding not only of the domain in general, but also of the role a concept plays in describing a specific domain.
This paper has also presented a structure of multiple shared ontologies for knowledge sharing. Although this is still on going research, we believe that such a structure has advantages over the others especially if considered in the context of an open environment such as the Internet. We believe that this kind of modularisation is the key to applications where intelligent agents (whose knowledge is represented by ontologies) interoperate dynamically, by agreeing on the vocabulary (and shared knowledge) which is closer to the conceptualisations of *only those agents which are involved in the interoparation* and not of all agents that can be potentially involved. We realise that we have not investigated in sufficient detail the issues related to building such structure in an efficient and cost effective manner, and the relationships existing within and between the ontologies composing the structure (both topics are future research directions that we will consider, see next section); but we think that we have laid the basis for future research.

7 Future work

Future research on ontology clusters concerns the relationships between and within ontologies, which need to be clarified with respect to previous work presented in the literature. Two candidate sets of relations have been identified, these are Borst's *ontology projections*: include and extend, include and specialise, include and map [2]; and Visser and Cui's *ontology relations*: subset/superset, extension, restriction, mapping [42]. Another issue emerging from this research is how knowledge sources (or agents), reach consensus on which cluster in the structure of multiple shared ontologies they have to join in order to achieve interoperation. This kind of consensus should be based on suitable similarity measure, that take into account the semantics of the concepts involved, and the semantics of their properties. There are no similarity functions of this type, that we are aware of, and it would be interesting to investigate complex similarity measures, such as those for symbolic objects [4]. We are particularly interested in investigating similarity functions that make use of the extra semantics provided by the conceptual metamodel, in a way analogous to the similarity measure presented in [29]. These kind of similarity functions usually provide a measure of the degree of similarity among different concepts, and not just a binary measure that indicates whether two concepts are similar or not.
From the viewpoint of the ontology conceptual metamodel, future work include understanding the kind of inferences and the reasoning mechanisms that are supported by the additional semantics included in the ontology metamodel. In order to support complex reasoning inferences, we will consider the implementation of the metamodel in some description logic's based language, which should provide the capabilities to perform the inferences. This model is also quite demanding to use, future work should concentrate also on identifying the kinds of applications that can benefit from the expressive power provided by this model.
In order to test the effectiveness of the conceptual metamodel, we are planning to include the metaproperties in tools to build ontolo-

gies such as WebOde [1] or Protégé [7].

Acknowledgements

The authors wish to thank Asunción Gómez-Pé. The PhD presented in this paper was funded by BT plc.

REFERENCES

[1] J.C. Arpírez, O. Corcho, M. Fernández-López, and A. Gómez-Pérez, 'WebODE: A scalable workbench for ontological engineering', in *Proceedings of the First International Conference on Knowledge Capture, K-CAP 2001*. ACM-Sigmod, (2001).

[2] P. Borst, *Construction of Engineering ontologies for knowledge sharing and reuse*, Ph.D. dissertation, Centre for Telematica and Information Technology, University of Twente, 1997.

[3] A.M. Collins and E.F. Loftus, 'A spreading-activation theory of semantic processing', *Psychological Review*, **82**, 407–425, (1975).

[4] F. Esposito, D. Malerba, V. Tamma, and H.-H. Bock, 'Classical resemblance measures', in *Analysis of Symbolic data. Exploratory methods for extracting statistical information from complex data*, eds., H.-H. Bock and E. Diday, volume 15 of *Studies in Classification, Data Analysis, and Knowledge Organisation*, 139–152, Springer-Verlag, Berlin, (2000).

[5] M. Fernández-López, A. Gómez-Pérez, and N. Guarino, 'The methontology & ontoClean merge', Technical report, OntoWeb special interest group on Enterprise-standards Ontology Environments, (2001).

[6] M. Fernández-López, A. Gómez-Pérez, A. Pazos-Sierra, and J. Pazos-Sierra, 'Building a chemical ontology using METHONTOLOGY and the ontology design environment', *IEEE Intelligent Systems and their applications*, **January/February**, 37–46, (1999).

[7] N. Fridman Noy, R. W. Fergerson, and M. A. Musen, 'The knowledge model of protege-2000: Combining interoperability and flexibility', in *Proceedings of the 12th EKAW Conference*, ed., R. Dieng, volume LNAI 1937, pp. 17–32, Berlin, (2000). Springer Verlag.

[8] N. Fridman Noy and M.A. Musen, 'SMART: Automated support for ontology merging and alignment', in *Proceedings of the 12th Workshop on Knowledge Acquisition, Modeling and Management (KAW)*, Banff, Alberta, Canada, (1999). University of Calgary.

[9] N. Fridman Noy and M.A. Musen, 'Anchor-PROMPT: Using non-local context for semantic matching', in *Proceedings of the IJCAI'01 workshop on ontologies and information sharing*, eds., A. Gómez-Pérez, M. Gruninger, H. Stuchenschmidt, and M. Uschold, (2001). http://www.semantic-translation.com/IJCAIwp/.

[10] M. Goldszmidt and J. Pearl, 'Qualitative probabilisties for default reasoning, belief revision, and causal modelling', *Artificial Intelligence*, **84**(1-2), 57–112, (1996).

[11] A. Gómez-Pérez, 'Knowledge sharing and reuse', in *The Handbook of Applied Expert Systems*, ed., J. Liebowitz, 10.1–10.36, CRC Press LLC, Boca Raton, FL, (1998).

[12] T. R. Gruber, 'A translation approach to portable ontology specifications', *Knowledge Acquisition*, **5**(2), 199–220, (1993).

[13] N. Guarino, 'Formal ontologies and information systems', in *Proceedings of FOIS'98*, ed., N. Guarino, Amsterdam, (1998). IOS Press.

[14] N. Guarino, M. Carrara, and P. Giaretta, 'An ontology of meta-level-categories', in *Principles of Knowledge representation and reasoning: Proceedings of the fourth international conference (KR94)*, pp. 270–280, San Mateo, CA, (1994). Morgan Kaufmann.

[15] E. Hirsch, *The concept of identity*, Oxford University Press, New York, 1982.

[16] I. Kant, *Critique of pure reason*, St. Martin's press, New York, 1965. Translation by N. Kemp Smith from *Kritik der reinen Vernunft*, 1787.

[17] R. Kowalski and M. Sergot, 'A logic-based calculus of events', *New Generation Computing*, **4**, 67–95, (1986).

[18] S.A. Kripke, *Naming and necessity*, Harvard University Press, Cambridge, Massachusetts, USA, 1980.

[19] O. Lassila and D. McGuinness, 'The role of frame-based representation on the semantic web', *Electronic Transactions on Artificial Intelligence (ETAI) Journal: area The Semantic Web*, **To appear**, (2001).

[20] D.B. Lenat, 'Cyc: a large-scale investment in knowledge infrastructure', *Communications of the ACM*, **38**(11), 33–38, (November 1995).

[21] D.K. Lewis, *Counterfactuals*, Blackwell, Oxford, 1993.

[22] E.J. Lowe, *Kinds of being. A study of individuation, identity and the logic of sortal terms*, Basil Blackwell, Oxford, UK, 1989.

[23] D.L. McGuinness, 'Conceptual modelling for distributed ontology environments', in *Proceedings of the Eighth International Conference on Conceptual Structures Logical, Linguistic, and Computational Issues (ICCS 2000)*, eds., B. Ganter and G.W. Mineau, volume LNAI 1867, (2000).

[24] D.L. McGuinness, R.E. Fikes, J. Rice, and S. Wilder, 'An environment for merging and testing large ontologies', in *Principles of Knowledge Representation and Reasoning. Proceedings of the seventh international conference (KR'2000)*, eds., A.G. Cohn, F. Giunchiglia, and B. Selman, pp. 483–493, San Francisco, CA, (2000). Morgan Kaufmann.

[25] G.A. Miller, 'Nouns in WordNet: a lexical inheritance system', *International Journal of Lexicography*, **3**(4), 245–264, (1990).

[26] A. Plantiga, *The nature of necessity*, Clarendon Library of logic and philosophy. Clarendon Press, New York, 1989.

[27] M.R. Quillian, 'Semantic memory', in *Semantic Information Processing*, ed., Marvin Minsky, 227–270, MIT Press, Cambridge, Massachusetts, (1968).

[28] R. Rada, H. Mili, E. Bicknell, and M. Blettner, 'Development and application of a metric on semantic nets', *IEEE Transactions on Systems, Man, and Cybernetics*, **19**(1), 17–30, (1989).

[29] M.A. Rodriguez and M.J. Egenhofer, 'Determining semantic similarity among entity classes from different ontologies', *IEEE transactions on knowledge and data engineering*, (2002). in press.

[30] E.H. Rosch, 'Cognitive representations of semantic categories', *Journal of Experimental Psychology: General*, **104**, 192–233, (1975).

[31] E.H. Rosch, 'Reclaiming concepts', *Journal of Consciousness Studies*, **6**(11-12), 61–77, (1999).

[32] J.R. Searle, *Intentionality*, Cambridge University Press, Cambridge, 1983.

[33] M.J.R. Shave, 'Ontological structures for knowledge sharing', *The new review of information networking*, **3**, 125–133, (1997).

[34] J.F. Sowa, *Conceptual Structures: Information Processing in Mind and Machine*, Addison-Wesley, Reading, MA, 1984.

[35] J.F. Sowa, *Knowledge Representation: Logical, Philosophical, and Computational Foundations*, Brooks Cole Publishing Co., Pacific Grove, CA, 2000.

[36] F. Steimann, 'On the representation of roles in object-oriented and conceptual modelling', *Data and Knowledge Engineering*, **35**, 83–106, (2000).

[37] R. Studer, V.R. Benjamins, and D. Fensel, 'Knowledge engineering, principles and methods', *Data and Knowledge Engineering*, **25**(1-2), 161–197, (1998).

[38] V.A.M. Tamma and T.J.M. Bench-Capon, 'An enriched knowledge model for formal ontological analysis', in *Proceedings of the international conference on formal ontology and information systems (FOIS'01)*, eds., C. Welty and B. Smith, New York, (2001). ACM press.

[39] V.A.M. Tamma and T.J.M. Bench-Capon, 'An ontology model to facilitate knowledge sharing in multi-agent systems', *Knowledge Engineering Review*, **To appear**, (2002).

[40] A. Tversky, 'Features of similarity', *Psychological Review*, **84**(4), 327–372, (1977).

[41] G. van Heijst, A.Th. Schreiber, and B.J. Wielinga, 'Using explicit ontologies in kbs development', *International Journal of Human-Computer Studies*, **45**, 184–292, (1997).

[42] P. Visser and Z. Cui, 'On accepting heterogeneous ontologies in distributed architectures', in *Proceedings of the ECAI'98 workshop on Applications of Ontologies and Problem-solving methods, Brighton, UK*, eds., P. Borst, V.R. Benjamins, A. Farquhar, and A. Gómez-Pérez, pp. 112–119, (1998).

[43] P.R.S. Visser, D.M. Jones, T.J.M. Bench-Capon, and M.J.R. Shave, 'Assessing heterogeneity by classifying ontology mismatches', in *Formal Ontology in Information Systems. Proceedings FOIS'98, Trento, Italy*, ed., N. Guarino, pp. 148–182. IOS Press, (1998).

[44] C. Welty and N. Guarino, 'Supporting ontological analysis of taxonomical relationships', *Data and knowledge engineering*, **39**(1), 51–74, (2001).

[45] D. Wiggins, *Identity and Spatio-Temporal continuity*, Basil Blackwell, Oxford, 1967.

System Descriptions

Ontological Issues in the Representation and Presentation of Mathematical Concepts

Armin Fiedler[1], **Andreas Franke**[1], **Helmut Horacek**[1], **Markus Moschner**[1], **Martin Pollet**[1] and **Volker Sorge**[2]

Abstract. Major purposes underlying the functionality of formal systems include reasoning services and presentation facilities, prominently their systematic coordination. An important role in this coordination is played by the ontology and its underlying organization principles exhibited in the systems' knowledge bases. We address this issue by presenting specific components of our proof development system ΩMEGA, which combines reasoning facilities for handling mathematical proofs with presentation capabilities in natural language. These components are the mathematical knowledge base of ΩMEGA and the linguistic knowledge base of the attached proof explanation system *P.rex*. We feature their ontological principles, modeling coverage, and their interoperability. Interfacing reasoning and presentation skills is crucial for increasing the quality in illustrating the results of formal inference systems.

1 INTRODUCTION

Major purposes underlying the functionality of formal systems include reasoning services and presentation facilities, in particular their systematic coordination. An important role in this coordination is played by the ontology and its underlying organization principles exhibited in the systems' knowledge bases, prominently the interoperability across several knowledge bases.

We address this issue by a case study, presenting specific components of our proof development system ΩMEGA [14]. ΩMEGA is a mathematical assistant tool that supports proof development at a user-friendly level of abstraction. The system combines interactive and automated proof construction in mathematical domains. Figure 1 illustrates the architecture of ΩMEGA: several independent modules are connected via the mathematical software bus MATHWEB-SB [5]. An important benefit is that MATHWEB modules can be distributed over the Internet and are accessible by other distant research groups as well. The core of ΩMEGA is the proof plan data structure $\mathcal{PDS}$ [3], the proof planner MULTI [11], the suggestion mechanism Ω-ANTS [2], and a hierarchy of mathematical theories, represented by a mathematical data base, which constitutes the basic mathematical ontology of the system.

Various heterogeneous external reasoning systems with complementary capabilities are integrated into ΩMEGA (see the right side of Figure 1). ΩMEGA interfaces systems such as computer algebra systems (CASs), higher- (HO) and first-order (FO) automated theorem proving systems (ATPs), constraint solvers (CSs), and model generators (MGs). Their use is twofold: they may provide a solution to a subproblem, or they may give hints for the control of the proof search. These systems are either connected directly or indirectly via proof transformation modules, in order to orchestrate their use, and to integrate their results. The output of the incorporated reasoning systems is translated and inserted as sub-proofs in ΩMEGA's central proof plan data structure, which maintains proof plans simultaneously at different levels of abstraction. This is beneficial for interfacing systems that operate at divergent levels of abstraction as well as for a human oriented display and inspection of a partial proof. From an ontological perspective, the mathematical theories in ΩMEGA provide some sort of normative ontology for mathematical concepts, to which concepts of external systems must be related.

User interaction is supported by the graphical user interface $\mathcal{LOUI}$ [15] and the proof explainer *P.rex* [4] (see the left side of Figure 1). The latter is a particular feature of ΩMEGA, since it provides proof explanations in natural language, in interactive and adaptive form. In order to be able to describe mathematical concepts in context, *P.rex* needs its own ontology, organized on the demands of natural language presentations.

In the following, we address the two major subsystems involved in ontological issues, the knowledge bases of ΩMEGA and *P.rex*, in dedicated sections. Thereby, we feature their divergent ontological principles and complementary modeling coverage. Next, we address their interoperability. Finally, we list case studies carried out with ΩMEGA and we sketch relations to other approaches to ontology.

2 THE REPRESENTATION COMPONENT

The organization of ΩMEGA's knowledge base is motivated by the following observations. The statement of a mathematical theorem can depend on the availability of a possibly large set of definitions of mathematical concepts, which in turn, may themselves depend on other concepts. Moreover, previously proven theorems or lemmas may be reused within the context of a proof. Going beyond pure representation purposes, a formal reasoning system needs access to other forms of knowledge, including inference steps, such as tactics, and information about control knowledge for automated reasoners

[1] Computer Science Department, Saarland University, P.O.Box 151150, D-66041 Saarbrücken, Germany
[2] School of Computer Science, University of Birmingham, Birmingham B15 2TT, UK

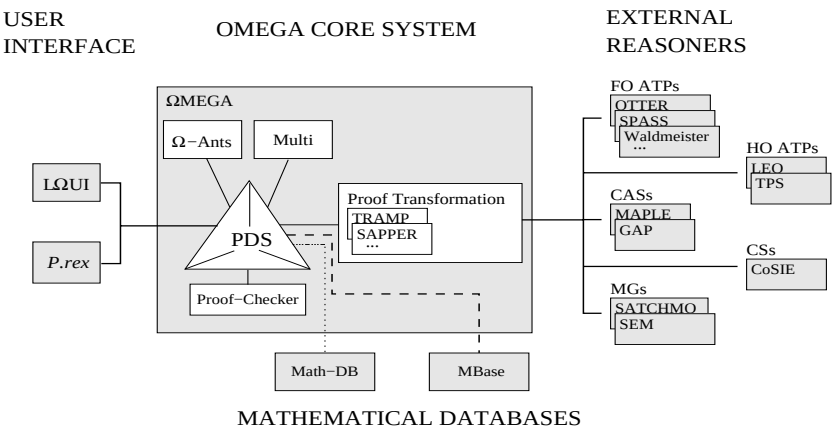

Figure 1. The architecture of the ΩMEGA proof assistant. Thin lines denote internal interfaces, thick lines denote communication via MATHWEB-SB. ΩMEGA has access to its local mathematical database (thin dotted line) and MBASE (thick dashed line).

and about human-oriented presentation knowledge. To support reasoning in such an environment, the main requirement is an efficient access to definitions and axioms, thereby avoiding redundancy in storing that information to ease maintenance. For meeting this demand, we have decided to use an inheritance network. It provides most effective percolation of information to build the core semantics of each mathematical object. In contrast, it ignores their meaning and interrelations other than componential ones, since this knowledge is irrelevant for the intended use.

In order to define a mathematical concept, which is a structured object with several properties, a number of options typically exist, depending on the subfield in question. The prototypical alternative lies between the use of strictly *minimal* properties in logical terms and *commonly used* ones. The latter are stronger than the former ones and might contain redundancies, but they may be better known in the field or preferred for some social reasons. Hence, it is logically inferable that these properties hold, once it is known that the minimal properties hold. Consequently, it may be possible to define a mathematical concept in several equivalent ways within the same theory, each of which may prove useful for reasoning purposes.

Let us consider a well-known example for which a number of alternative, yet equivalent, definitions exist, the notion of a *group*. A group is usually defined as a nonempty set together with an operation, where the operation is closed and associative for the elements of this set. Moreover, there exists a unit element, and for each elements of this set an inverse element. While this definition is the most common one, it is not minimal: for instance we can replace the existence of the unit element or of inverse elements with respective weaker properties that only a left unit element or left inverse elements exist. Still we would arrive at a definition of a group, equivalent to the first. Moreover, we can replace the postulation of the unit element and the inverses entirely by the single axiom: *For all* $a, b \in G$ *exist* $x, y \in G$ *such that* $a \circ x = b$ *and* $y \circ a = b$.

We can also define the notion of a group via larger, less concrete algebraic structures. Hereby we have several possible ways to arrive at a definition as is outlined in Figure 2.

Let G be a nonempty set and let $\circ$ be a binary operation on G. The following assertions are equivalent:

- $(G, \circ)$ is a group.
- $(G, \circ)$ is a monoid and every element of $a \in G$ has an inverse.
- $(G, \circ)$ is a loop and $\circ$ is associative.
- $(G, \circ)$ is both a quasi-group and a semi-group.

The three variants except to the first one, each correspond to one path leading to *Group* in Figure 2, where the link coming from *Loop* adds the property of associativity, and the link coming from *Monoid* adds the property of the existence of inverses. In order to establish these equivalent representations within the system in a justified manner, the equivalences entailed must be proved.

To support this task, we are currently replacing the knowledge base by the system MBASE [6], which is a distributed mathematical knowledge base designed to make storage and access through queries efficient. The specification of correspondences on the ontological level is to be carried out interactively and supported by theorem proving devices.

In order to store and manipulate these kinds of information, MBASE distinguishes several categories of information objects, on which the structure of the underlying data base model is grounded:

- Definitions for associating meanings to symbols in terms of already defined ones.
- Assertions, which are logical sentences, including axioms, theorems, conjectures and lemmas, distinguished according to pragmatic or proof-theoretic status. item Proofs, encapsulating the actual proof objects in various formats, including formal, informal, system-dependent proof scripts, and even natural language formats.
- Examples, due to their importance in mathematical practice.
- Theories, which allow grouping of mathematical objects and knowledge and the introduction of inheritance between theories.
- Inference steps, in form of system-dependent programming code, as basic calculus rules and compound steps.
- Human-oriented (technical) knowledge, such as names for theorems, and specifications for notation and symbol handling.

The data base model contains some relations between objects of these kinds onto which inheritance is made. These include

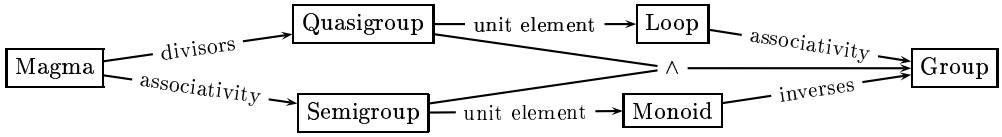

Figure 2. Constructing groups via more general algebras

- Definition-entailment, to relate defined symbols to others and, ultimately, to symbols marked as primitive.
- Depends-on/Local-in, which specify dependency and locality information for primary knowledge items. The relation makes explicit the use of symbols in a definition or assertion, and the application of lemmas in a proof.
- Theory-inheritance, which specifies the organization of mathematical subdomains and the associated inheritance of their properties.

The purpose of the categories described before is to ensure correctness in the process of building mathematics on computers and at the same time ease this process. The introduced entities should help to construct proofs but at the same time avoid additional efforts in their administration. Definitions as explicit entities allow for the abbreviation of terms and formulas. At the same time it has to be ensured that the definition functionality will not introduce inconsistencies. The explicit hierarchy of theories allows for grouping of mathematical knowledge and helps the user to keep an overview. On the other hand one has to decide on how to structure the formalizations so that it will be useful for the proof construction in theories that inherit these theories. As described for the equivalent definitions of the concept *Group* there seems to be no best formalization that covers every aspect and the rigidity of the theory hierarchy has to be equalized by mechanisms as the development graph [9], which allows us to manage a later change of underlying formalizations.

The given categorization is not specific to ΩMEGA but holds as well for other proof development systems which maintain a library of mathematical knowledge. The systems can differ in the following aspects:

- finer classification of objects, e.g. recursive definitions in COQ [16],
- the language used for terms and formulas, e.g typed lambda calculus in ΩMEGA or set theory in MIZAR [13],
- the available functionality, e.g. theory interpretations as mechanism for the reuse of theorems in IMPS [10],
- the basic logical calculus and the formalism to build more complex inference steps.

3 THE PRESENTATION COMPONENT

P.rex is a proof explainer attached to ΩMEGA, which is responsible for expressing a mathematical proof in terms of natural language text, interleaved with formulas. A particular capability of *P.rex* is to explain a proof step at different levels of abstraction, initially at the most abstract level that the user is assumed to know. The system reacts flexibly to questions and requests. While the explanation is in progress, the user can interrupt *P.rex* anytime, if the current explanation is not satisfactory. *P.rex* analyzes the user's interaction and enters into a clarification dialog when needed to identify the reason why the explanation was not satisfactory and re-plans a better explanation, for example, by switching to another level of abstraction. During the presentation process, *P.rex* constructs a *discourse structure tree* to organize the utterances to be produced. Each utterance is represented by a tree, called *Text Structure*, that encodes its linguistic specification (cf. [8] for details).

The specifications in a proof at some level of abstraction, which *P.rex* is supposed to convert into a natural language presentation, contain references to mathematical concepts. For presentation purposes, these are organized in terms of *semantic categories*. The main role of the *semantic categories* is to provide vocabularies, which specify type restrictions for nodes of the Text Structure. They define how separate Text Structures can be combined, and ensure that the planner only build expressible Text Structures. For instance, if tree A should be expanded at node n by tree B, the resulting type of B must be compatible to the type restriction attached to n. Following Panaget [12], however, we split the type restrictions into two orthogonal dimensions: the ideational dimension in terms of the *Upper Model* [1], and the *hierarchy of textual semantic categories* to be discussed below. As in other conceptual hierarchies, the relation between the top level node and its immediate successors is realized as a specialization, rather than as an instantiation. Since the meaning of this relation is never communicated, this inaccuracy is tolerable.

Technically speaking, the Text Structure in *P.rex* is a tree recursively composed of kernel subtrees or composite subtrees: An atomic *kernel subtree* has a head at the root and arguments as children, representing basically a predicate/argument structure. *Composite subtrees* can be divided into two subtypes: the first has a special *matrix* child and zero or more *adjunct* children and represents linguistic hypotaxis, the second has two or more *coordinated* children and stands for parataxis.

Each node is typed both in terms of the Upper Model and the hierarchy of textual semantic categories. The Upper Model is a domain-independent property inheritance network of concepts that are hierarchically organized according to how they can be linguistically expressed. Figure 3 shows a fragment of the Upper Model in *P.rex*. For every domain of application, domain-specific concepts must be identified and placed as an extension of the Upper Model. In the domain of mathematics, most domain-specific concepts (formally: one-place predicates) are placed under the concept *non-conscious thing*, except to some real-world concepts, which appear in mathematical subtheories for puzzles. Moreover, domain properties appear under *discrete place relations*. The organization of these items is driven by specialization relations and by type restrictions of the fillers of relations. Note that these principles are complementary to those of the mathematical knowledge base, which leads to differently structured (typically flatter) hierarchies.

The hierarchy of textual semantic categories is also a domain-independent property inheritance network. The con-

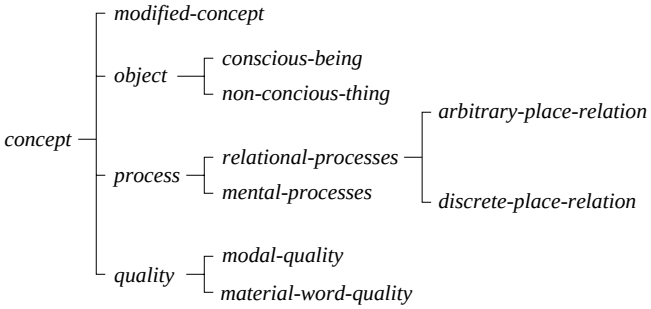

Figure 3. A Fragment of the Upper Model in *P.rex*

cepts are organized in a hierarchy based on their textual realization. For example, the concept *clause-modifier-rankingI* is realized as an adverb, *clause-modifier-rankingII* as a prepositional phrase, and *clause-modifier-embedded* as an adverbial clause. Figure 4 shows a fragment of the hierarchy of textual semantic categories.

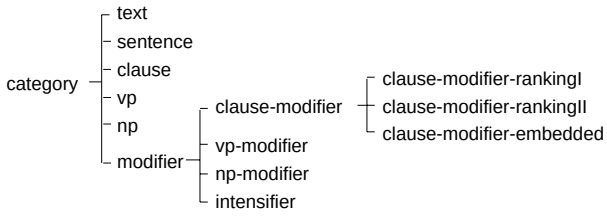

Figure 4. A Fragment of the Hierarchy of Textual Semantic Categories in *P.rex*

The equivalences in defining groups or similar mathematical objects, as exposed in the previous section, are not reflected by the linguistic knowledge bases. Therefore, building an alternative representation, which may be motivated by various presentation goals, has to be carried out on the basis of the proof representation prior to verbalization, which is a matter of future work. In contrast, *P.rex* is able to produce different phrasings for some mathematical properties, such as associativity, the adjective 'associative' or the noun 'associativity', enabled through the interplay of the Upper Model, the lexicon, and the textual semantic categories. This distinction is not reflected by the mathematical knowledge base.

4 INTEROPERABILITY

When comparing the organization principles of MBase and the Upper Model of *P.rex*, it is interesting to see that the reasoning purposes motivating the organization in MBase do not require specialization in their hierarchical structuring, which is quite common for other ontologies. In the Upper Model, however, the way these specializations are set up is oriented on presentation purposes rather than on mathematical properties. Conversely, the precise logical definitions percolated through the inheritance network expressing the semantics of the mathematical concepts and relations are not accessible to the Upper Model.

Through these definitions extending the Upper Model, the mathematical concepts are in some sense re-represented, with links to their counterparts in the mathematical knowledge base. The associated maintenance effort is unavoidable, since for each mathematical notion newly modeled in the mathematical theory part, the corresponding counterpart on the linguistic side must be appropriately integrated in the linguistic knowledge bases. Only for those cases, where a mathematical concept requires no linguistic knowledge other than reference by its name, we use a short-cut for handling interoperability between mathematical and linguistic knowledge bases. Mathematical concepts of this kind are mapped onto a catch-all concept in the linguistic knowledge base, and reference by name to this concept is done by passing through its name from the mathematical knowledge base. The main advantage of this realization lies in some degree of independency, that is, extending the mathematical knowledge base and re-running the system without adapting the linguistic knowledge base accordingly is possible, if the resulting (in some cases, temporary) limitation in the presentation quality is acceptable. Maintaining this short-cut will be impossible when the extensions described below will be addressed.

Apart from this basic integration, purposes of presentation impose some additional demands that cannot be met adequately by the inheritance network and by the Upper Model in its present state. Those which can be met easier by system extensions are the following:

- Items preferred for referring expression need to be marked. For example, the terms *Group* and *Semigroup* are more common than *Monoid*, which identifies a similar algebraic structure. The more common terms should be preferred in descriptions even though this may require extended descriptions.
- Conceptual equivalences must be made explicit to avoid redundancy in presentations. As demonstrated in Section 2, for example, various possibilities to define a group in algebraic terms, and switching between definitions might easily result in a strange rephrasing of an obvious equivalence. For reasoning purposes, the equivalence is established by an inference step or, in more complicated cases, by a subproof.
- Additional, highly special conceptual definitions that have no relevance for reasoning purposes may improve the presentation capabilities significantly. Typically, axioms that are expressed as compound or nested rules are good sources for building such definitions, which relate to subexpressions that appear in that axiom and are likely to be described as intermediate results in proofs.

The techniques described so far mainly support the coordination of static knowledge originating from heterogeneous sources for problem solving purposes. For presentation purposes, there are two fundamental shortcomings of the present representation, which severely limit presentation capabilities:

- The connection between mathematical objects and Upper Model objects is too simple, since it is restricted to one-to-one correspondences. This is meaningfully applicable to domain terms, but not to some relations and inference rules.
- The dynamic knowledge (e.g., proofs, examples) is represented too coarse-grained and on a superficial level only, which limits variations for presenting it.

These shortcomings are as fundamental as they are deliberate, since the complexity of the design and development tasks for MBase is high enough. For future extensions, these are good candidates for linking more linguistically oriented tools.

5 CASE STUDIES AND AVAILABILITY

The ΩMEGA system is available on the Internet at `http://www.ags.uni-sb.de/~omega`. It has been evaluated in multiple case studies, which have been carried out for varying purposes motivated by theorem-proving issues; as a consequence, ΩMEGA's knowledge bases contain representation fragments of a variety of subdomains, including:

ε-δ-proofs: Limit theorems and assertions about functions and sequences and continuity of functions.

Automatic classification of residue-class structures: Proofs of properties of residue classes and about isomorphisms between residue class structures.

Interactive proof planning: A combination of Ω-ANTS and MULTI was used to support the interactive exploration of residue classes and to prove theorems about properties of group homomorphisms in student exercises.

Set theory: Ω-ANTS has been applied with ND rules, an automated theorem prover, and a model generator to prove or disprove set equations.

Group theory: Equivalence of different group definitions, uniqueness of the unit element and inverse element.

The software components referred to in this paper are further described at the following websites:

- ΩMEGA: `http://www.ags.uni-sb.de/~omega/soft/omega`
- *P.rex*: `http://www.ags.uni-sb.de/~prex`
- MBASE: `http://www.mathweb.org/mbase`

6 CONCLUSION AND DISCUSSION

In comparison to other domains, high quality representations are required, so that they can only be produced in a handcrafted manner. Apart from that quality aspect, building ontologies for the domain of mathematics can in some sense be considered less difficult than a similar task for another domain:

- Vagueness does not play a role at all.
- There is a high degree of agreement about the domain concepts; discrepancies merely concern alternative representation variants, formats, and conventions.
- Although the domain as a whole is very large, it can be reasonably well broken down into subdomains of manageable size.

In some sense, the domain is atypical, since the domain objects are completely artificial from the reasoning perspective, with the exception of puzzle subdomains. From the presentation perspective, however, the representation purposes are not much different from other domains. They do differ in terms of richness and degrees of variety, which is more limited in our domain in comparison to narratives. Consequently, our Upper Model is merely a copy of the one used by Penman [1], oriented on the purpose of expressibility. The main emphasis in our approach is establishing a basic interoperability.

Altogether, we have shown that representation and presentation purposes in the domain of mathematical can be met by distributed knowledge bases, with different organization principles, complementary coverage, and minimal linking, which reduces the maintenance effort. Not surprisingly, the present, simple design, imposes limitations on the overall functionality. In general, shortcomings in the presentation part raise demands that can be met best by extensions in the mathematical representation component or by some module that manipulates these representations. We have identified several of these shortcomings, which we will address in future work. When doing this, we expect aspects of formal ontologies, such as identity and unity [7], to become relevant for our presentation part as well.

REFERENCES

[1] J. A. Bateman, R. T. Kasper, J. D. Moore, and R. A. Whitney, 'A general organization of knowledge for natural language processing: the Penman upper model', Technical report, University of Southern California, Information Science Institute, (1990).

[2] C. Benzmüller and V. Sorge, 'Ω-ANTS – An open approach at combining Interactive and Automated Theorem Proving', in *Proc. of Calculemus-2000*, eds., M. Kerber and M. Kohlhase. AK Peters, (2001).

[3] L. Cheikhrouhou and V. Sorge, '$\mathcal{PDS}$ — A Three-Dimensional Data Structure for Proof Plans', in *Proc. of ACIDCA'2000*, (2000).

[4] A. Fiedler, 'Dialog-driven adaptation of explanations of proofs', in *Proc. of the 17th International Joint Conference on Artificial Intelligence (IJCAI)*, ed., B. Nebel, pp. 1295–1300, Seattle, WA, (2001). Morgan Kaufmann.

[5] A. Franke and M. Kohlhase, 'System description: MATHWEB, an agent-based communication layer for distributed automated theorem proving', in *Proc. of CADE-16*, ed., H. Ganzinger, number 1632 in LNAI. Springer, (1999).

[6] A. Franke and M. Kohlhase, 'System description: MBASE, an open mathematical knowledge base', in *Proc. of CADE-17*, ed., D. McAllester, number 1831 in LNAI. Springer, (2000).

[7] N. Guarino and C. Welty, 'Identity, unity, and individuality: Towards a formal toolkit for ontological analysis', in *Proc. of ECAI-2000*. IOS Press, (2000).

[8] X. Huang and A. Fiedler, 'Proof verbalization as an application of NLG', in *Proc. of the 15th International Joint Conference on Artificial Intelligence (IJCAI)*, ed., M.E. Pollack, pp. 965–970, Nagoya, Japan, (1997). Morgan Kaufmann.

[9] D. Hutter, 'Management of change in structured verification', in *Proceedings Automated Software Engineering (ASE-2000)*. IEEE Press, (2000).

[10] The IMPS online theory library. Internet interface at `http://imps.mcmaster.ca/theories/theory-library.html`.

[11] E. Melis and A. Meier, 'Proof Planning with Multiple Strategies', in *Proc. of CL-2000*, volume 1861 of *LNAI*. Springer, (2000).

[12] F. Panaget, 'Using a textual representational level component in the context of discourse or dialogue generation', in *Proc. of the 7th International Workshop on Natural Language Generation*, pp. 127–136, Kennebunkport, ME, (1994). INLG.

[13] Piotr Rudnicki, 'An overview of the mizar project', in *Proceedingsof the 1992 Workshop on Types and Proofs as Program s*, pp. 311–332, (1992).

[14] J. Siekmann, C. Benzmüller, V. Brezhnev, L. Cheikhrouhou, A. Fiedler, A. Franke, H. Horacek, M. Kohlhase, A. Meier, E. Melis, M. Moschner, I. Normann, M. Pollet, V. Sorge, C. Ullrich, C.-P. Wirth, and J. Zimmer, 'Proof development with ΩMEGA', in *Proc. of CADE-18*, Copenhagen, Denmark, (2002). forthcoming.

[15] J. Siekmann, S. Hess, C. Benzmüller, L. Cheikhrouhou, A. Fiedler, H. Horacek, M. Kohlhase, K. Konrad, A. Meier, E. Melis, M. Pollet, and V. Sorge, '$\mathcal{LOUI}$: $\mathcal{L}$ovely ΩMEGA $\mathcal{U}$ser $\mathcal{I}$nterface', *Formal Aspects of Computing*, **11**, 326–342, (1999).

[16] Coq Development Team, *The Coq Proof Assistant Reference Manual*, INRIA. see `http://coq.inria.fr/doc/main.html`.

SEEKing Knowledge in Legacy Information Systems to Support Interoperability

Joachim Hammer, Mark Schmalz, William O'Brien[¥], Sangeetha Shekar and **Nikhil Haldevnekar**
Dept. of Computer & Information Science & Engineering
University of Florida
Gainesville, FL 32605, U.S.A.

Abstract. The SEEK project (Scalable Extraction of Enterprise Knowledge) is developing methodologies to overcome the problems of assembling knowledge resident in numerous legacy information systems by enabling rapid connection to, and privacy-constrained filtering of, legacy data and applications with little programmatic setup. In this report we outline our use of data reverse engineering and code analysis techniques to automatically infer as much as possible the schema and semantics of a legacy information system. We illustrate the approach using an example from our construction supply chain testbed.

1 MOTIVATION

We are developing methodologies and algorithms to facilitate discovery and extraction of enterprise knowledge from legacy sources. These capabilities are being implemented in a toolkit called SEEK (Scalable Extraction of Enterprise Knowledge). SEEK is being developed as part of a larger, multi-disciplinary research project to develop theory and methodologies in support of computerized decision and negotiation support across a network of firms (general overview in [6]). SEEK is not meant as a replacement for wrapper or mediator development toolkits. Rather, it complements existing tools by providing input about the contents and structure of the legacy source that has so far been supplied manually by domain experts. This streamlines the process and makes wrapper development scalable.

Figure 1 illustrates the need for knowledge extraction tools in support of wrapper development in the context of a supply chain. There are many firms (principally, subcontractors and suppliers), and each firm contains legacy data used to manage internal processes. This data is also useful as input to a project level decision support tool. However, the large number of firms working on a project makes it likely that there will be a high degree of physical and semantic heterogeneity in their legacy systems. This implies practical difficulties in connecting firms' data and systems with enterprise-level decision support tools. It is the role of the SEEK toolkit to help establish the necessary connections with minimal burden on the underlying firms, which often have limited technical expertise. The SEEK wrappers shown in Fig. 1 are wholly owned by the firm they are accessing and hence provide a safety layer between the source and end user. Security can be further enhanced by deploying the wrappers in a secure hosting infrastructure at an ISP, for example, as shown in the figure.

We note that SEEK is not intended to be a general-purpose data extraction tool: SEEK extracts a narrow range of data and knowledge from heterogeneous sources. Current instantiations of SEEK are designed to extract the limited range of information needed by these process models to support project optimization.

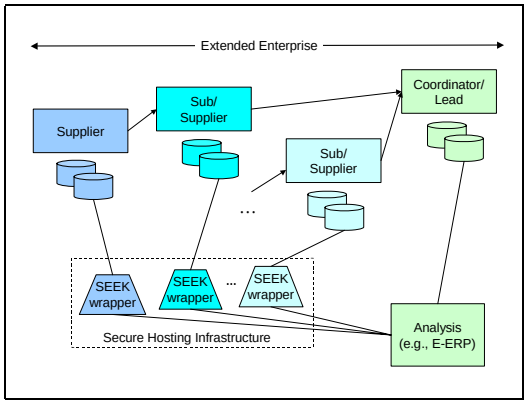

Figure 1: Using the SEEK toolkit to improve coordination in extended enterprises.

2 SEEK APPROACH TO KNOWLEDGE EXTRATCION

SEEK applies Data Reverse Engineering (DRE) and Schema Matching (SM) processes to legacy database(s), to produce a source wrapper for a legacy source. The source wrapper will be used by another component (for the analysis component in Figure 1) wishing to communicate and exchange information with the legacy system.

First SEEK generates a detailed description of the legacy source, including entities, relationships, application-specific meanings of the entities and relationships, business rules, data formatting and reporting constraints, etc. We collectively refer to this information as *enterprise knowledge*. The extracted enterprise knowledge forms a knowledgebase that serves as input for subsequent steps. In particular, DRE connects to the underlying DBMS to extract schema information (most data sources support some form of Call-Level Interface such as JDBC). The schema information from the database is semantically enhanced using clues extracted by the semantic analyzer from available application code, business reports, and, in the future, perhaps other electronically available information that may encode business data such as e-mail correspondence, corporate memos, etc. It has been our experience (through visits with representatives from the construction and

[¥] Rinker School of Building Construction, University of Florida, Gainesville, FL 32634-6134

manufacturing domains) that such application code exists and can be made available electronically. Second, the semantically enhanced legacy source schema must be mapped into the domain model (DM) used by the application(s) that want(s) to access the legacy source. This is done using a schema mapping process that produces the mapping rules between the legacy source schema and the application domain model. In addition to the domain model, the schema mapper also needs access to the domain ontology (DO) describing the model.

Finally, the extracted legacy schema and the mapping rules provide the input to the wrapper generator (not shown), which produces the source wrapper. In this paper, we focus on our implementation of the DRE algorithm.

3 Data Reverse Engineering

Data reverse engineering (DRE) is defined as the application of analytical techniques to one or more legacy data sources to elicit structural information (e.g., term definitions, schema definitions) from the legacy source(s) in order to improve the database design or produce missing schema documentation. So far in SEEK, we are applying DRE to relational databases only. However, since the relational model has only limited semantic expressability, in addition to the schema, our DRE algorithm generates an E/R-like representation of the entities and relationships that are not explicitly defined in the legacy schema (but which exist implicitly). Our approach to data reverse engineering for relational sources is based on existing algorithms by Chiang [1, 2] and Petit [8]. However, we have improved their methodologies in several ways, most importantly to reduce the dependency on human input and to eliminate some of the limitations of their algorithms (e.g., consistent naming of key attributes, legacy schema in 3-NF).

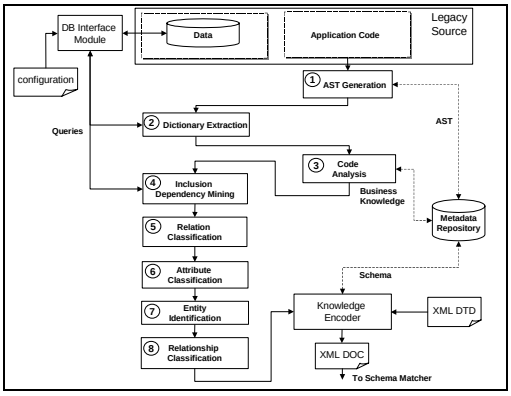

Figure 2: Conceptual overview of the DRE algorithm.

Our DRE algorithm is divided into schema extraction and semantic analysis, which operate in interleaved fashion. An overview of the two algorithms, which are comprised of eight steps, is shown in Figure 2. In addition to the modules that execute each of the eight steps, the architecture in Figure 3 includes three support components: the configurable *Database Interface Module* (upper-right hand corner), which provides connectivity to the underlying legacy source. Note that this component is the ONLY source-specific component in the architecture: in order to perform knowledge extraction from different sources, only the interface module needs to be changed. The *Knowledge Encoder* (lower right-hand corner) represents the extracted knowledge in the form of an XML document so that it can be shared with other components in the SEEK architecture (e.g., the semantic matcher). The *Metadata Repository* is internal to DRE and used to store intermediate run-time information needed by the algorithms including user input parameters, the abstract syntax tree for the code (e.g., from a previous invocation), etc.

We now highlight each of the eight steps and related activities outlined in Figure 3 using an example from our construction supply chain testbed. For a detailed description of our algorithm, refer to [3]. For simplicity, we assume without lack of generality or specificity that only the following relations exist in the MS-Project application, which will be discovered using DRE (for a description of the entire schema refer to [5]):

```
MSP-Project [PROJ_ID, ...]
MSP-Availability[PROJ_ID, AVAIL_UID, ...]
MSP-Resources [PROJ_ID, RES_UID,  ...]
MSP-Tasks [PROJ_ID, TASK_UID,  ...]
MSP-Assignment [PROJ_ID, ASSN_UID,  ...]
```

In order to illustrate the code analysis and how it enhances the schema extraction, we refer the reader to the following C code fragment representing a simple, hypothetical interaction with the MS Project database.

```
char *aValue, *cValue;
int flag = 0;
int bValue = 0;
EXEC SQL SELECT A,C INTO :aValue, :cValue
FROM Z WHERE B = :bValue;
if (cValue < aValue)
       {   flag = 1; }
printf("Task Start Date %s ", aValue);
printf("Task Finish Date %s ", cValue);
```

Step 1: AST Generation

We start by creating an Abstract Syntax Tree (AST) shown in Figure 3. The AST will be used by the semantic analyzer for code exploration during step 3. Our objective in AST generation is to be able to associate "meaning" with program variables. Format strings in input/output statements contain semantic information that can be associated with the variables in the input/output statement. This program variable in turn may be associated with a column of a table in the underlying legacy database.

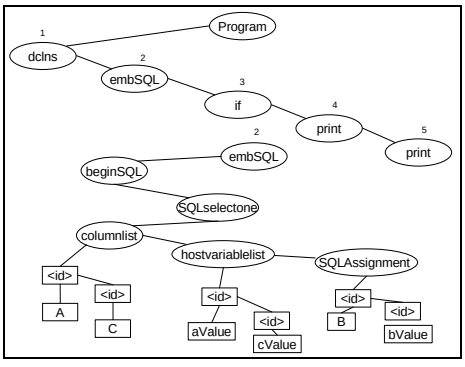

Figure 3: Application-specific code analysis via AST decomposition and code slicing. The direction of slicing is backwards (forward) if the variable in question is in an output (resp. input or declaration) statement.

Step 2. Dictionary Extraction.

The goal of step 2 is to obtain the *relation* and *attribute names* from the legacy source. This is done by querying the data dictionary, stored in the underlying database in the form of one or more system tables. Otherwise, if primary key information cannot be retrieved directly from the data dictionary, the algorithm passes the set of candidate keys along with predefined "rule-out" patterns to the code analyzer. The code analyzer searches for these patterns in the application code and eliminates those attributes from the candidate set, which occur in the rule-out pattern. The rule-out patterns, which are expressed as SQL queries, occur in the application code whenever programmer expects to select a SET of tuples. If, after the code analysis, not all primary key can be identified, the reduced set of candidate keys is presented to the user for final primary key selection.

Result. In the example DRE application, the following relations and their attributes were obtained from the MS-Project database:

```
MSP-Project [PROJ_ID, ...]
MSP-Availability[PROJ_ID, AVAIL_UID, ...]
MSP-Resources [PROJ_ID, RES_UID,  ...]
MSP-Tasks [PROJ_ID, TASK_UID,  ...]
MSP-Assignment [PROJ_ID, ASSN_UID,  ...]
```

Step 3: Code Analysis

The objective of step 3, code analysis, is twofold: (1) augment entities extracted in step 2 with domain semantics, and (2) identify business rules and constraints not explicitly stored in the database, but which may be important to the wrapper developer or application program accessing the legacy source. Our approach to code analysis is based on code analysis, which includes slicing [4] and pattern matching [7].

The first step is the *pre-slicing*. From the AST of the application code, the pre-slicer identifies all the nodes corresponding to input, output and embedded SQL statements. It appends the statement node name, and identifier list to an array as the AST is traversed in pre-order. For example, for the AST in Figure 3, the array contains the following information depicted in Table 1. The identifiers that occur in this data structure maintained by the pre-slicer form the set of slicing variables.

Table 1: Information maintained by the pre-slicer.

Node number	Statement	Text String (for print nodes)	Identifiers	Direction of Slicing
2	embSQL (Embedded SQL node)	-----	aValue cValue	Backwards

The code slicer and analyzer, which represent steps two and three respectively, are executed once for each slicing variable identified by the pre-slicer. In the above example, the slicing variables that occur in SQL and output statements are aValue and cValue. The direction of slicing is fixed as backwards or forwards depending on whether the variable in question is part of a output (backwards) or input (forwards) statement. The slicing criterion is the exact statement (SQL or input or output) node that corresponds to the slicing variable.

During code slicing sub-step we traverse the AST for the source code and retain only those nodes that have an occurrence of the slicing variable in sub-tree. This results in a reduced AST, which is shown in Fig. 4.

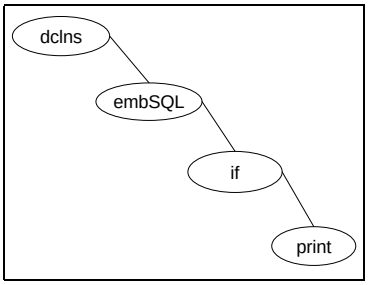

Figure 4: Reduced AST.

During the analysis sub-step, our algorithm extracts the information shown in Table 2, while traversing the reduced AST in pre-order.

1. If a *dcln* node is encountered, the data type of the identifier can be learned.
2. *embSQL* contain the mapping information of identifier name to corresponding column name and table name in the database.
3. *Printf/scanf* nodes contain the mapping information from the text string to the identifier. In other words we can extract the 'meaning' of the identifier from the text string.

Table 2: Information inferred during the analysis sub-step.

Identifier Name	Meaning	Possible Business Rule

aValue	Task Start Date	if (cValue < aValue)
		{
		}

cValue	Task Finish Date	if (cValue < aValue)
		{
		}

Data type	Column Name in Source	Table Name in Source
Char * => string	A	Z
Char * => string	C	Z

The results of analysis sub-step are appended to a result report file. After the code slicer and analyzer have been invoked on every slicing variable identified by the pre-slicer, the results report file is presented to the user. The user can base his decision of whether to perform further analysis based on the information extracted so far. If the user decides not to perform further analysis, code analysis passes control to the inclusion dependency detection module.

It is important to note, that we identify enterprise knowledge by matching templates against code fragments in the AST. So far, we have developed patterns for discovering business rules which are encoded in loop structures and/or conditional statements and mathematical formulae, which are encoded in loop structures and/or assignment statements. Note, the occurrence of an assignment statement itself does not necessarily indicate the presence of a mathematical formula, but the likelihood increases significantly if the statement contains one of the "slicing variables."

Step 4. Discovering Inclusion Dependencies.

After extraction of the relational schema in step 2, the goal of step 4 is to identify constraints to help classify the extracted relations, which represent both the real-world entities and the relationships among them. This is done using inclusion dependencies (INDs), which indicate the existence of inter-relational constraints including class/subclass relationships.

Let A and B be two relations, and X and Y be attributes or a set of attributes of A and B respectively. An inclusion dependency A.X << B.Y denotes that a set of values appearing in A.X is a subset of B.Y. Inclusion dependencies are discovered by examining all possible subset relationships between any two relations A and B in the legacy source.

Without additional input from the domain expert, inclusion dependencies can be identified in an exhaustive manner as follows: for each pair of relations A and B in the legacy source schema, compare the values for each non-key attribute combination X in B with the values of each candidate key attribute combination Y in A (note that X and Y may be single attributes). An inclusion dependency B.X<<A.Y may be present if:

1. X and Y have same number of attributes.
2. X and Y must have pair wise domain compatibility.
3. B.X ≥ A.Y

In order to check the subset criteria (3), we have designed the following generalized SQL query templates, which are instantiated for each pair of relations and attribute combinations and run against the legacy source:

```
C1 =                    C2 =
SELECT count (*)        SELECT count (*)
FROM R1                 FROM R2
WHERE U NOT IN          WHERE V NOT IN
    (SELECT V                   (SELECT U
     FROM R2);                  FROM R1);
```

If C1 is zero, we can deduce that there may exist an inclusion dependency R1.U << R2.V; likewise, if C2 is zero there may exist an inclusion dependency R2.V << R1.U. Note that it is possible for both C1 and C2 to be zero. In that case, we can conclude that the two sets of attributes U and V are equal.

The worst-case complexity of this exhaustive search, given N tables and M attributes per table (NM total attributes), is O(N2M2). However, we reduce the search space in those cases where we can identify equi-join queries in the application code (during semantic analysis). Each equi-join query allows us to deduce the existence of one or more inclusion dependencies in the underlying schema. In addition, using the results of the corresponding count queries we can also determine the "direction" of the dependencies. This allows us to limit our exhaustive searching to only those relations not mentioned in the extracted queries.

Result: Inclusion dependencies are as follows:

1 MSP_Assignment[Task_uid,Proj_ID] << MSP_Tasks [Task_uid,Proj_ID]
2 MSP_Assignment[Res_uid,Proj_ID] << MSP_Resources[Res_uid,Proj_ID]
3 MSP_Availability [Res_uid,Proj_ID] << MSP_Resources [Res_uid,Proj_ID]
4 MSP_Resources [Proj_ID] << MSP_Project [Proj_ID]
5 MSP_Tasks [Proj_ID] << MSP_Project [Proj_ID]
6 MSP_Assignment [Proj_ID] << MSP_Project [Proj_ID]
7 MSP_Availability [Proj_ID] << MSP_Project [Proj_ID]

The last two inclusion dependencies are removed since they are implicitly contained in the inclusion dependencies listed in lines 2, 3 and 4 using the transitivity relationship.

Step 5. Classification of the Relations.

When reverse-engineering a relational schema, it is important to understand that due to the limited expressability of the relational model, all real-world entities are represented as relations irrespective of their types and role in the model. The goal of this step is to identify the different "types" of relations, some of which correspond to actual real-world entities while others represent relationships among them.

In this step all the relations in the database are classified into one of four types – strong, regular, weak or specific. Identifying different relations is done using the primary key information obtained in step 2 and the inclusion dependencies from step 4. Intuitively, a strong entity-relation represents a real-world entity whose members can be identified exclusively through its own properties. A weak entity-relation represents an entity that has no properties of its own that can be used to identify its members. In the relation model, the primary keys of weak entity-relations usually contain primary key attributes from other (strong) entity-relations. Both regular and specific relations are relations that represent relationships between two entities in the real world (rather then the entities themselves). However, there are instances when not all of the entities participating in an (n-ary) relationship are present in the database schema (e.g., one or more of the relations were deleted as part of the normal database schema evolution process). While reverse engineering the database, we identify such relationships as special relations.

Result:

Strong Entities: `MSP_Projects`
Weak Entities: `MSP_Resources, MSP_Tasks, MSP_Availability`
Regular Relationship: `MSP-Assignment`

Step 6. Classification of the Attributes.

We classify attributes as (a) PK or FK (from DRE-1 or DRE-2), (b) Dangling or General, or (c) Non-Key (rest).

Result: Table 3 illustrates attributes obtained from the example legacy source.

Table 3. Example of attribute classification from MS-Project legacy source.

	PKA	DKA	GKA	FKA	NKA
MS-Project	Proj_ID				All Remaining Attributes
MS-Resources	Proj_ID	Res_uid			
MS-Tasks	Proj_ID	Task_uid			
MS-Availability	Proj_ID	Avail_uid		Res_uid+ Proj_ID	
MS-Assignment	Proj_ID		Assn_uid	Res_uid+ Proj_ID, Task_uid + Proj_ID	

Step 7. Identify Entity Types.

Strong (weak) entity relations obtained from step 5 are directly converted into strong (resp. weak) entities.

Result: The following entities were classified:

Strong entities:

`MSP_Project` with `Proj_ID` as its key.

Weak entities:

`MSP_Tasks` with `Task_uid` as key and `MSP_Project` as its owner.

`MSP_Resources with Res_uid as key and MSP_Project as its owner.`

`MSP_Availability` with `Avail_uid` as key and `MSP_Resources` as owner.

Step 8. Identify Relationship Types.

The inclusion dependencies discovered in step 4 form the basis for determining the relationship types among the entities identified above. This is a two-step process:

1. Identify relationships present as relations in the relational database. The relation types (regular and specific) obtained from the classification of relations (Step 5) are converted into relationships. The participating entity types are derived from the inclusion dependencies. For completeness of the extracted schema, we may decide to create a new entity when conceptualizing a specific relation.

   The cardinality between the entities is M:N.

2. Identify relationships among the entity types (strong and weak) that were not present as relations in the relational database, via the following classification.

   ∉# *IS-A relationships* can be identified using the PKAs of strong entity relations and the inclusion dependencies among PKAs. The cardinality of the IS-A relationship between the corresponding strong entities is 1:1.

   ∉# *Dependent relationship*: For each weak entity type, the owner is determined by examining the inclusion dependencies involving the corresponding weak entity-relation. The cardinality of the dependent relationship between the owner and the weak entity is 1:N.

   Aggregate relationships: If the foreign key in any of the regular and specific relations refers to the PKA of one of the strong entity relations, an aggregate relationship is identified. The cardinality is either 1:1 or 1:N.

   ∉# *Other binary relationships*: Other binary relationships are identified from the FKAs not used in identifying the above relationships. If the foreign key contains unique values, the cardinality is 1:1, else the cardinality is 1:N.

Result:

We discovered 1:N binary relationships between the following weak entity types:

Between `MSP_Project` and `MSP_Tasks`
Between `MSP_Project` and `MSP_Resources`
Between `MSP_Resources` and `MSP_Availabilty`

Since two inclusion dependencies involving `MSP_Assignment` exist (i.e., between `Task` and `Assignment` and between `Resource` and `Assignment`), there is no need to define a new entity. Thus, `MSP_Assignment` becomes an M:N relationship between `MSP_Tasks` and `MSP_Resources`.

At the end of Step 8, DRE has extracted the following schema information from the legacy database:

∉# Names and classification of all entities and attributes.

∉# Primary and foreign keys.

∉# Data types.

∉# Simple constraints (e.g., unique) and explicit assertions.

∉# Relationships and their cardinalities.

∉# Business rules

A conceptual overview of the extracted schema is represented by the entity-relationship diagram shown in Figure 5 (business rules not shown), which is an accurate representation of the information in encoded in the original MS Project schema.

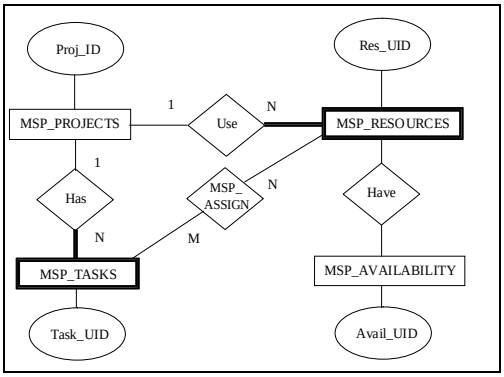

Figure 5: E/R diagram representing the extracted schema.

4 STATUS AND FUTURE WORK

We have manually tested our approach for a number of scenarios and domains (including construction, manufacturing and health care) to validate our knowledge extraction algorithm and to estimate how much user input is required. In addition, we have also conducted experiments using nine different database applications that were created by students during course projects. The experimental results so far are encouraging: the DRE algorithm was able to reverse engineer all of the sample legacy sources encountered so far. When coupled with semantic analysis, human input is reduced compared to existing methods. Instead the user is presented with clues and guidelines that lead to the augmentation of the schema with additional semantic knowledge.

The SEEK prototype is being extended using sample data from a large building construction project on the University of Florida campus in cooperation with the manager, Centex Rooney Inc., and several subcontractors or suppliers. This data testbed will support much more rigorous testing of the SEEK toolkit. Other plans for the SEEK toolkit are:

∉# Develop a formal representation for the extracted knowledge.

∉# Develop a matching tool capable of producing mappings between two semantically related yet structurally different schemas. Currently, schema matching is performed manually, which is a tedious, error-prone, and expensive process.

∉# Integrate SEEK with a wrapper development toolkit to determine if the extracted knowledge is sufficiently rich semantically to support compilation of legacy source wrappers for our construction testbed.

ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under grant numbers CMS-0075407 and CMS-0122193. The authors also thank Dr. Raymond Issa for his valuable comments and feedback on a draft of this paper.

REFERENCES

[1] R. H. Chiang, "A knowledge-based system for performing reverse engineering of relational database," *Decision Support Systems*, **13**, pp. 295-312, 1995.

[2] R. H. L. Chiang, T. M. Barron, and V. C. Storey, "Reverse engineering of relational databases: Extraction of an EER model from a relational database," *Data and Knowledge Engineering*, **12**:1, pp. 107-142., 1994.

[3] J. Hammer, M. Schmalz, W. O'Brien, S. Shekar, and N. Haldavnekar, "Knowledge Extraction in the SEEK Project," University of Florida, Gainesville, FL 32611-6120, Technical Report TR-0214, June 2002.

[4] S. Horwitz and T. Reps, "The use of program dependence graphs in software engineering," in *Proceedings of the Fourteenth International Conference on Software Engineering*, Melbourne, Australia, 1992.

[5] Microsoft Corp., "Microsoft Project 2000 Database Design Diagram", http://www.microsoft.com/office/project/prk/2000/Download/VisioHTM/P9_dbd_frame.htm.

[6] W. O'Brien, R. R. Issa, J. Hammer, M. S. Schmalz, J. Geunes, and S. X. Bai, "SEEK: Accomplishing Enterprise Information Integration Across Heterogeneous Sources," *ITCON - Journal of Information Technology in Construction*, 2002.

[7] S. Paul and A. Prakash, "A Framework for Source Code Search Using Program Patterns," *Software Engineering*, **20**:6, pp. 463-475, 1994.

[8] J.-M. Petit, F. Toumani, J.-F. Boulicaut, and J. Kouloumdjian, "Towards the Reverse Engineering of Denormalized Relational Databases," in *Proceedings of the Twelfth International Conference on Data Engineering (ICDE)*, New Orleans, LA, pp. 218-227, 1996.

Finding and Integration of Information - A Practical Solution for the Semantic Web -

Ubbo Visser and **Gerhard Schuster**[1]

Abstract. If we believe the numerous publications concerning intelligent approaches for better information retrieval from the WWW the Semantic Web is already alive. However, the nature of most of the approaches is more theoretical. One major outcome of the research being undertaken over the last few years in the area of artificial intelligence for the Semantic Web is the benefit of using ontologies for content-based information retrieval. This led to a number of systems that provide user interfaces and intelligent reasoning services to access and integrate information sources (e.g. Ontobroker, SHOE, OntoSeek, BUSTER). This paper deals with a practical solution for finding and integrating information from the Web. Since some of the ideas of our BUSTER system are already known we focus on two issues: we introduce the Comprehensive Source Description (CSD), a necessary description for information sources that allows extra services such as integration or translation and a new feature that allows a combined search for concepts at a certain location, introducing the *concept@location* query. We discuss implementation issues and provide an example for better understanding.

1 INTRODUCTION

The Internet as de facto biggest information source electronically available consists of a vast amount of data, which are mainly loosely structured. Mostly, these data belong to proprietary systems, which are not build for interoperability in the first place. With the comprehensive networking it is nowadays possible to link the items in the network together. Thus, there is a need for tools that are able to find, access, and integrate the information sources. The main obstacles are schematic and semantic heterogeneity problems, which are thoroughly discussed in various papers [6, 14, 25]. Over the last decade several approaches with regard to intelligent information integration have been proposed (e.g. IM, SIMS, OBSERVER, COIN; see [25] for an overview). The majority of these systems provide representation mechanisms for ontology-based content explication. The systems mainly use some kind of description logics (e.g. OIL). The main reason behind this is the option to explicitly describe concepts of an application domain using a language that provides formal semantics. Lately, this general approach of using ontology-based systems for information integration has been widely accepted [8].

Ontologies became a popular research topic in the 90ies and are still the focus of researchers in the artificial intelligence area. There is still a need for more fundamental research in various areas: the role of ontologies, acquisition of ontologies, semantic mapping and translation to name only a few topics. This may only be one reason why ontology-based systems are mainly theoretical approaches with some prototypical front-ends. A new article in a trendy computer magazine [27] states that there are numerous publications with respect to the Semantic Web but there are only a few applications available. There is a need for practical solutions. The BUSTER system is a contribution for this demand as it provides means for ontology-based search and integration.

In this paper we discuss the BUSTER approach focussing on the description of information sources and describe our prototypical implementation. We introduce a new feature of the system, namely the option to search for concepts using a terminological reasoning service and to search for locations using a spatial reasoning service. A combination of both leads to a new query type *concept@location*, e.g. ”Are there land cover sources available that cover Lower-Saxony?” or searching for ”suppliers for product X in region Y”.

2 APPROACH

The Bremen University Semantic Translator for Enhanced Retrieval (BUSTER), a middleware based also on ontologies, has been developed at the Center for Computing Technologies. BUSTER is based on the hybrid ontology approach, i.e. it can access more than one ontology and integrate them. The only restriction is that there is a common vocabulary the ontologies are based on. Schuster and Stuckenschmidt [21] describe a method that leads to a common vocabulary using known but domain dependent thesauri.

The concept view of the system is shown in figure 1. It shows the query phase on the right hand side and the acquisition phase on the left hand side. Since the description of information sources with meta data is crucial we focus on the *Comprehensive Source Descriptions* (CSD), located at the site of the data source or service, and formalized in XML/RDF format. A thorough description about concept of BUSTER can be found in [22].

Comprehensive Source Description

In order to describe existing data metadata have to be used. Hence, we have to find an eligible language for the description. Over the last decade numerous meta data formats have emerged (e.g. Dublin Core, ISO/TC211). A good overview about existing meta information systems can be found in [23]. Since we are not dependent on any specific domain, in fact we would like to use a general way to describe the data, we use the Dublin Core Element Set, version 1.1 as a de facto basis for our CSD. The definitions utilize a formal standard for the description of metadata elements. The authors claim that the formalization helps to improve consistency with other metadata communi-

[1] Center for Computing Technologies, Universität Bremen, Universitätsallee 21-23, D-28359 Bremen, Germany, email: {visser|schuster}@tzi.de

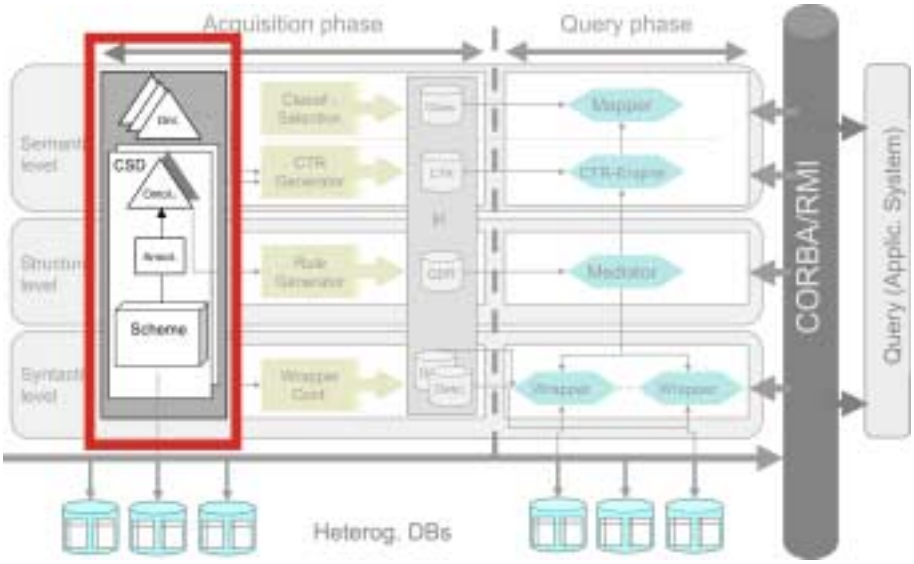

Figure 1. BUSTER: concept view

ties and enhances the clarity, scope, and internal consistency of the Dublin Core metadata element definitions.

However, some of the given elements are not sophisticated enough in their expressivity (e.g. the relation element) or lack formal semantics (e.g. description element). Thus, there is a need for additional qualifiers for those elements, which are described in a language that provides formal semantics (e.g. DAML, OIL, SHIQ). We can use this kind of description logics to encode additional features. We use the RDF(S) syntax if possible to ensure a wide acceptance with respect to accessibility and usability. We then refer to explicit ontologies available on the WWW. The following Dublin Core elements are refined for our CSD:

- Coverage: Since there is no further distinction between spatial and temporal coverage, this element has to be refined.
  - Spatial: The recommended best practice from DCMI is to select a value from a controlled vocabulary and that, where appropriate, named places or time periods be used in preference to numeric identifiers such as sets of coordinates or date ranges. Examples are *DCMI Point* to describe a point in space using its geographic coordinates, *ISO 3166* a code for the representation of names of countries, *DCMI Box* that identifies a region of space using its geographic limits. The last recommendation is *TGN*, the GETTY Thesaurus of Geographic Names (see http://shiva.pub.getty.edu/tgn_browser/). We decided on the latter because the use of place names is more intuitive and therefore more valuable with respect to users on the WWW.
  - Temporal: The recommend best practice here is to use one of the two following encoding schemes: *DCMI Period*, a specification of the limits of a time interval, and *W3C-DTF*, the W3C encoding rules for dates and times - a profile based on ISO 8601 (see also: http://www.w3.org/TR/NOTE-datetime). We use the latter since the main reason to have this CSD is to describe information sources on the WWW.
- Description: Description may include but is not limited to: an abstract, table of contents, reference to a graphical representation of content or a free-text account of the content. The semantics of this kind of representation are limited with regards to machine readable meaning of the content. Hence, we restrict the description to a formal description logic, namely DAML+OIL or SHIQ. The vocabulary used to describe this A-Boxes has to be one of the vocabularies used in the ”relation” element.
- Relation: The qualifiers that refine the relation element as recommended by DCMI is limited. Therefore, we need to extend these qualifiers by references that also point to ontologies, gazetteers or thesauri. A relation is described as a XML namespace describing the URI of the corresponding vocabulary and a prefix to mark terms from this vocabulary.
- Subject: The qualifiers recommended by DCMI for the subject element contain common lists of keyword from various sources (e.g. the Library of Congress Subject Headings, Medical Subject Headings, Universal Decimal Classification). In BUSTER, we use the subject element accordingly, it remains a list of significant keywords to describe the information source but the keywords have to be chosen from a controlled vocabulary referred by the relation element.
- Rights: Despite the intellectual property rights we also have to consider access rights for special user groups. In the moment, there is no further specification.

Figure 2 shows an extract from a typical CSD, a CSD for a data set concerning land use in Lower-Saxony, Germany in this particular case. We only show the relevant parts according to the refined elements mentioned above. The subject contains links to a ”topic-area” described in the general CSD ontology and some concepts concerning the content of the topic-area described in the GEMET ontology. The ”description” element consists of two additional properties of that information source (a) the fact the data set consist of a Bessel-ellipsoid from 1841, which is described in a geodesic ontology, and (b) the meaning of the attributes of the underlying relational table. One might think that this is additional modeling effort for no good

Figure 2. Extract of a typical Comprehensive Source Description (CSD)

reason but we are now able to enable additional services such as automatic translation processes between catalogue systems as described in [17].

Based on the metadata provided by the CSDs and appropriate qualitative terminological (conceptual ontologies) and spatial models (spatial ontologies), BUSTER supports integrated queries of the type *concept@location*. These type of queries are described in the next section.

3 IMPLEMENTATION

The prototype of the BUSTER is based on an open server-client architecture, and can be divided into two main parts: the so-called BUSTER-cluster on the server side and a BUSTER client.

3.1 Architecture

BUSTER clients can be started as local applications or as java applets in a standard browser supporting Java Swing. The BUSTER client provides an ontology-driven user interface to specify queries and to present the results of the retrieval. Additional services such as automatic translation process will be made available dependent on the result and if applicable. The communication between the clients and the cluster is implemented via Remote Method Invocation (RMI).

The BUSTER cluster comprises several modules relevant for intelligent querying and semantic translation purposes: a BUSTER server, a database for CSDs and available domains, a web server, and spatial and logical reasoning modules (see figure 3). Examples for the latter available on the WWW are the FaCT system provided by the University of Manchester [13] and the RACER system provided by the University of Hamburg [10]. These modules are within the BUSTER cluster to fit the minimum requirements for terminology and spatial queries, but its open architecture allows to use arbitrary services for reasoning, translation or other tasks if needed.

An Apache web server provides the platform for the applets. The server handles client queries depending on the users selection. It controls the process of the query *(concept@location)* by retrieving domain specific information from a SQL-database via JDBC interface, downloading distributed CSDs and knowledge bases, and triggering reasoning services within or outside the BUSTER cluster.

3.2 Queries

Once an information source has been annotated with all the information needed, complex queries can be made to the BUSTER system. BUSTER is based on terminological ontologies that have been modelled in advance. The system allows two different types of queries, a terminological and a spatial query. We introduced the query *concept@location* however, it is possible to submit a terminological query alone without spending time on the spatial part. Also it is possible to submit a spatial query on its own.

3.2.1 Terminological Queries

The terminological query can be divided into two parts, namely a simple concept query and a defined concept query.

Simple Concept Query The user chooses one terminology (ontology) depending on current domain, e.g. ”installation supplies”. These terminologies are registered at the BUSTER server. The user can then select on one of the concepts of the taxonomy that fits his query best (e.g. ”installation pipe”). The BUSTER server receives the query and integrates the known terminologies for the current domain by loading them into the connected reasoner. This is possible, because every terminology is annotated with a common vocabulary (hybrid approach, see section 2). After re-classification, all sub-concepts (children) of the query concepts form the result.

Defined Concept Query According to the domain, the user chooses a query-template provided by the BUSTER server. This template contains attributes (slots) and values (filler) from the common vocabulary. The user-interface is ontology-driven, which simply means that the available attributes and fillers are automatically loaded and presented dynamically. This way the user can’t make a mistake, e.g. using unknown terms. The user defines his query by selecting

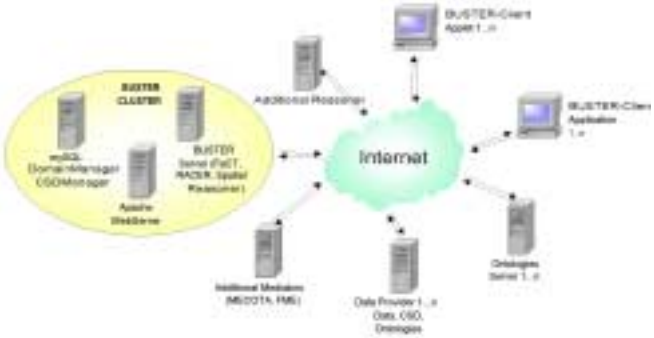

Figure 3. BUSTER: system architecture

Figure 4. Example for a defined concept and spatial query

reasonable values for the given attributes. 'Yes' specifies the occurrence of the related filler, 'no' prohibits the occurrence and 'n/a' is chosen, if the value does not matter. Figure 4 shows an example of a defined concept query. The user is interested in information about the land cover of Lower-Saxony, a state of Germany. He chooses an appropriate query-template "DataObject", which provides attributes and values for the definition of information sources like databases. He selects the value "landcover" for the attribute "topic". He is not interested in information sources that deal with tourism or statistics. No statements are made about the other values. The filled query-template is translated into a logical term. During the query process all CSDs related to the current domain are parsed for the subject-tag. Each subject references to a namespace, which points to an ontology that contains a concept description of the subject term. These ontologies are then downloaded from ontology servers available on the WWW, are merged with the defined concept query and transferred into available inference machines. After re-classification, all sub-concepts (children) of the query concepts form the result.

In case of a simple concept query, the user has to choose a specific terminology. This makes the query simpler to understand for a user, but it assumes that the user knows at least one terminology or concept from the hierarchy. Simple concept queries are fast, but not always expressive enough. To overcome these problems one could use the defined concept query. On the base of the given common vocabulary the user is able to define a concept that fits his vision of a concrete concept. A defined query is more complex to build, but it is much more unrestricted.

3.2.2 Spatial Queries

A user-friendly and, from a cognitive perspective, sound method to specify spatial queries as well as to index data sources and services is the use of placenames. Placenames are typically organized in gazetteers [12, 18]. Schlieder et al. [19, 20] propose an extension to gazetteers in the form of placename structures based on qualitative spatial models. A placename structure can be seen as a hierarchical tree, where the nodes of the tree represent well known name descriptors for geographic features, and the edges reflect their binary part-of relations. These models, or spatial ontologies, use graph representations of hierarchically organized polygonal tessellations as a basis to reason about the spatial relevance of one placename with respect to another. In a qualitative spatial model tree leaves corresponding to nodes of the used connection graph represent the tessellation (see figure 5). Spatial relevance, a combined evaluation of partonomic and neighborhood relations between placenames, is computed by calculating the horizontal and vertical (or hierarchical) graph-theoretical distances.

In BUSTER the user is able to select a specific spatial ontology to initialize a spatial query. In our example the spatial model of Germany is selected. By selecting a placename (e.g. "Niedersachsen"), the user defines the target area of the spatial query. Using the selected spatial ontology, the spatial reasoner integrated in the BUSTER server evaluates the query and computes a list of placenames that are spatially relevant to the target placename. The user is able to parameterize the query by adjusting weight sliders for horizontal and vertical relevance. The example query is configured to find only information sources that are vertical relevant, like sources annotated with "Niedersachsen" or "Hannover" (district of lower-saxony).

3.2.3 Combined Spatio-Terminological Queries

BUSTER combines both lists, the list of relevant concepts, and the list of spatially relevant placenames, into one database query. This database query is applied to the BUSTER CSD database. The result is a weighted list of data sources and services matching both the terminological and the spatial query. Figure 6 shows the result of our combined query example. The data source found is an excel sheet with data classified by the Corine land cover nomenclature [4]. Relevant information as well as applicable services from the retrieved CSD are presented. As for the additional service, the user can choose the context translation from Corine to ATKIS [1].

4 Related work

Ontobroker [5] is a well known approach that relies on a single ontology for a group of web users. Therefore, both the data providers and the users have complete access and knowledge to all the concepts described in the ontology. Ontobroker is tailored to homogenous Intranet applications, e.g. for knowledge management within companies. Ontobroker relies on F-Logic and offers therefore advanced inference possibilities. KAON, the KArlsruhe ONtology and Semantic Web infrastructure [3] provides a general three-tier conceptual architecture, which consists of a client layer, a management layer, and a storage layer. The idea of this infrastructure based on RDF and ontologies is to provide services for advanced Semantic Web applications. KAON is a growing family of tools for engineering, discovery, management, and visualisation of ontologies.

OntoSeek [9] is designed for content-based information retrieval from online yellow pages and product catalogues. The retrieval techniques are based on lexical conceptual graphs and large linguistic ontologies (Sensus, WordNet). The basic architecture is similar however, BUSTER uses JAVA applets running in an arbitrary browser on

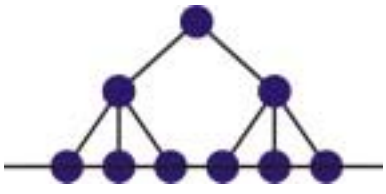

Figure 5. Example for placename structure. The nodes on thee base line represent the tessellation whereas the nodes above represent the placenames

Figure 6. Result for a defined and spatial query

an arbitrary OS. The main difference lies in the expressiveness capabilities of the ontology representation language. In BUSTER we use a more expressive description logic to describe concepts. Another major difference is the possibility to use further services such as a translation service between catalogue systems (if applicable) or a combined search for concepts at locations.

The SHOE Search Tool [11] allows a user to access a SHOE knowledge base by submitting structured queries. This query corresponds to the defined query in our system. The result is presented in a separate window and the user can doubleclick the found URIs to open the corresponding documents. The main differences with respect to BUSTER are the use of ontologies, the query service and other features such as translation services. In Buster, we can use several ontologies for one query, we are able to combine terminological and spatial search, and we can adopt additional mediators for further services shown in the result window.

The Information Manifold (IM) system [15] implements a client with a knowledge base for organizing and querying Internet information sources. The knowledge base contains a rich domain model that enables the description of properties of the information sources. The language used is based on a combination of Horn rules and concepts from the CLASSIC description logic [2]. In contrast to BUSTER IM is based on a single ontology approach using one global ontology. This approach can be applied to integration problems where all information sources to be integrated provide nearly the same view on a domain. If one information source has a different view on a domain, e.g. by providing another level of granularity, finding the minimal ontology commitment becomes a difficult task [7]. Another difference is the restriction of the IM system to only use database sources whereas BUSTER is also able to process other information sources such as XML-based sources.

A major difference between BUSTER and all other mentioned systems is the ability of BUSTER to combine both terminological reasoning and spatial reasoning.

5 CONCLUSION

We proposed a practical solution for finding information sources that have been annotated with metadata and offering additional services for processing the underlying data. We introduced the Comprehensive Source Description (CSD), a necessary description for information sources that allows extra services such as integration or translation. We also proposed a new feature that allows a combined search for concepts at a certain location, introducing the *concept@location* query.

We have seen that *interoperability between terminologies* is possible if we use the hybrid approach [25]. With our proposed practical solution we claim that *system interoperability* is now also feasible. The CSDs are flexible enough to annotate information sources providing additional knowledge to offer extra services. One important extra service could be a translation between different catalogue systems, which we already implemented and introduced elsewhere [17]. A definite drawback is the fact that there is an additional modeling effort. We think that there is a need for automatic annotation facilities. Promising new approaches are Text-To-Onto [16] or the MESA tool [24] for the construction of ontologies. The open architecture of our approach allows the use of additional mediators such as the Feature Manipulation Engine or MECOTA [26].

We think that the *concept@location* query with the included options to regulate both the spatial and the thematic distances are valuable for the Semantic Web. More than 80% of all the data available have an spatial context as we know and we think that our approach is a promising step in the right direction.

REFERENCES

[1] AdV, *Amtliches Topographisch-Kartographisches Informationssystem ATKIS*, Landesvermessungsamt NRW, Bonn, 1998.

[2] Ronald Brachman, Deborah McGuiness, Peter Patel-Schneider, Lori Resnik, and Alexander Borgida, 'Living with classic: When and how to use a kl-one-like language', in *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, ed., John Sowa, 401–456, Morgan Kaufman, San Mateo, CA, (1991).

[3] Stefan Decker, Michael Erdmann, Dieter Fensel, and Rudi Studer, 'Ontobroker: Ontology based access to distributed and semi-structured information.', in *Proceedings of Database Semantics 8*, ed., R. Meersman, Semantic Issues in Multimedia Systems., pp. 351–369. Kluwer Academic Publisher, Boston, (1999).

[4] EEA, 'Corine land cover', technical guide, European Environmental Agency, ETC/LC, European Topic Centre on Land Cover, (1997-1999 1997-1999).

[5] Dieter Fensel, Stefan Decker, M. Erdmann, and Rudi Studer, 'Ontobroker: The very high idea', in *11. International Flairs Conference (FLAIRS-98)*, Sanibal Island, USA, (1998). AAAI Press.

[6] Cheng Hian Goh, *Representing and Reasoning about Semantic Conflicts in Heterogeneous Information Sources*, Phd-thesis, MIT, 1997.

[7] Thomas R. Gruber, 'Toward principles for the design of ontologies used for knowledge sharing?', *International Journal of Human Computer Studies*, **43**(5/6), 907–928, (1995).

[8] Michael Grüninger and Mike Uschold, 'Ontologies and semantic integration', in *Software Agents for the Warfighter*, Institute for Human and Machine Cognition (IHMC), University of West Florida, (2002). in preparation.

[9] Nicola Guarino, Claudio Masolo, and Guido Vetere, 'Ontoseek: Content-based acess to the web', *IEEE Intelligent Systems*, **14**(3), 70–80, (1999).

[10] Volker Haarslev and Ralf Möller, 'High performance reasoning with very large knowledge bases', in *International Joint Conferences on Artificial Intelligence (IJCAI)*, ed., Bernhard Nebel, volume 1, pp. 161–166, Seattle, WA, (2001). Morgan Kaufman.

[11] Jeff Heflin and James Hendler, 'A portrait of the semantic web in action', *IEEE Intelligent Systems*, **16**(2), 54–59, (2001).

[12] Linda L. Hill, 'Core elements of digital gazetteers: placenames, categories, and footprints', in *ECDL 2000*, eds., J. Borbinha and T. Baker, Research and Advanced Technology for Digital Libraries, pp. 280–290, Lisbon, Portugal, (2000).

[13] I. Horrocks, 'Fact and ifact', in *Proceedings of the International Workshop on Description Logics (DL'99)*, eds., P. Lambrix, A. Borgida, M. Lenzerini, R. M ö ller, and P. Patel-Schneider, 133–135, CEUR-Workshop Proceedings at http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS, (1999).

[14] Won Kim and Jungyun Seo, 'Classifying schematic and data heterogeinity in multidatabase systems', *IEEE Computer*, **24**(12), 12–18, (1991). problem classification of semantic heterogeneity.

[15] T. Kirk, A. Y. Levy, Y. Sagiv, and D. Srivastava, 'The Information Manifold', in *Information Gathering from Heterogeneous, Distributed Environments*, eds., C. Knoblock and A. Levy, Stanford University, Stanford, California, (1995).
[16] Alexander Maedche and Steffen Staab, 'Ontology learning for the semantic web', *IEEE Intelligent Systems*, **16**(2), 72–79, (2001).
[17] Holger Neumann, Gerhard Schuster, Heiner Stuckenschmidt, Ubbo Visser, and Thomas Vögele, 'Intelligent brokering of environmental information with the buster system', in *International Symposium Informatics for Environmental Protection*, eds., Lorenz M. Hilty and Paul W. Gilgen, volume 30 of *Umwelt-Informatik Aktuell*, pp. 505–512, Zürich, Switzerland, (2001). Metropolis.
[18] W.-F. Riekert, 'Erschließung von fachinformationen im internet mit hilfe von thesauri und gazetteers', in *Management von Umweltinformationen in vernetzten Umgebungen, 2nd workshop HMI*, eds., C. Dade and B. Schulz, Nürnberg, (1999).
[19] Christoph Schlieder, Thomas Vögele, and Ubbo Visser, 'Qualitative spatial representation for information retrieval by gazetteers', in *Conference of Spatial Information Theory COSIT*, volume 2205 of *Spatial Information Theory: Foundations of Geographic Information Science*, pp. 336–351, Morrow Bay, CA, (2001). Springer.
[20] Christoph Schlieder and Thomas Vögele, 'Indexing and browsing digital maps with intelligent thumbnails', in *accepted at Spatial Data Handling 2002 (SDH'02)*, Ottawa, Canada, (2002). Springer. to appear.
[21] Gerhard Schuster and Heiner Stuckenschmidt, 'Building shared terminologies for ontology integration', in *Künstliche Intelligenz (KI)*, pp. 105–123, Wien, (2001).
[22] Heiner Stuckenschmidt, Holger Wache, Thomas Vögele, and Ubbo Visser, 'Enabling technologies for interoperability', in *Workshop: Information Sharing: Methods and Applications at the 14th International Symposium of Computer Science for Environmental Protection*, eds., Ubbo Visser and Hardy Pundt, volume 20, pp. 35–46, Bonn, (2000). TZI.
[23] Ubbo Visser, Heiner Stuckenschmidt, Holger Wache, and Thomas Vögele, 'Using environmental information efficiently: Sharing data and knowledge from heterogeneous sources', in *Environmental Information Systems in Industry and Public Administration*, eds., Claus Rautenstrauch and Susanne Patig, 41–73, IDEA Group, Hershey, USA & London, UK, (2001).
[24] Holger Wache, Thorsten Scholz, Helge Stieghahn, and B. König-Ries, 'An integration method for the specification of rule-oriented mediators', in *International Symposium on Database Applications in Non-Traditional Environments (DANTE'99)*, eds., Yahiko Kambayashi and Hiroki Takakura, pp. 109–112, Kyoto, Japan, (1999).
[25] Holger Wache, Thomas Vögele, Ubbo Visser, Heiner Stuckenschmidt, Gerhard Schuster, Holger Neumann, and Sebastian Hübner, 'Ontology-based integration of information - a survey of existing approaches', in *IJCAI-01 Workshop: Ontologies and Information Sharing*, eds., Asuncion Gómez Pérez, Michael Grüninger, Heiner Stuckenschmidt, and Mike Uschold, pp. 108–117, Seattle, WA, (2001).
[26] Holger Wache, 'Towards rule-based context transformation in mediators', in *International Workshop on Engineering Federated Information Systems (EFIS 99)*, eds., S. Conrad, W. Hasselbring, and G. Saake, Kühlungsborn, Germany, (1999). Infix-Verlag.
[27] Cai Ziegler, 'Deux ex machina', *c't - magazin für computer technik*, **2002**(6), 132–137, (2002).

Towards a Modularized Semantic Web

Raphael Volz, Daniel Oberle
Institute AIFB,
University of Karlsruhe,
76128 Karlsruhe, Germany

Alexander Maedche
FZI Forschungszentrum Informatik
University of Karlsruhe,
76131 Karlsruhe, Germany

ABSTRACT

Modularization is an established principle in software engineering. It has also been considered as a core principle when developing the World Wide Web. Along the same lines, the Semantic Web has to be based on modularization principles. This paper provides a first step into a modularized Semantic Web. It provides an elaborated and carefully evaluated view on existing technologies for naming, referring and modularization in the Web. Based on this analysis we propose means to import and include (parts) of RDF models by extending the RDF(S) meta-model, introducing new primitives for modularity.

1. INTRODUCTION

One general principle of powerful software systems is that they are built of many elements. Thus, when designing a system, the features of a system should be broken into relatively loosely bound groups of relatively closely bound features. Power comes from the interplay between the different elements. This interplay results in essential interdependencies and increases the ability to reuse and modify. Hence, future changes and consecutive testing can be limited to the relevant module. This will allow other people to independently change other parts at the same time. Modular design hinges on the simplicity and abstract nature of the interface definition between the modules. Notably, modularity was one of the core design goals for the World Wide Web.[1]. Along the same lines, the Semantic Web will not consist of neat ontologies that expert AI researchers have carefully constructed. Instead of a few large, complex, consistent ontologies that great numbers of users share, one will see a great number of small ontological elements consisting largely of pointers to each other [1].

We agree with this view and carefully evaluate existing technologies for naming, referring and modularization in the Web. We show that these technologies do not suffice for the task of building a truly modular Semantic Web. Based on our analysis we propose means to support this vision by extending the RDF meta-model, introducing new primitives for modularity. Namely, means for import and inclusion of (parts) of other RDF models. Additionally, we present an architectural setting for tools implementing this kind of modularization and finally give a set of engineering guidelines for building modularized Semantic Web applications on the basis of these novel means.

The structure follows the outlined procedure, thus section 2 provides an elaborated overview and evaluation of technologies relevant for modularization. Section 3 presents requirements for a modular Semantic Web and crafts an extension of the RDF meta-model to reflect these requirements. Section 4 provides reference applications using modularized ontologies and the aforementioned engineering guidelines for modularized Semantic Web applications from an ontology building perspective. Before we conclude and recapitulate our contribution in section 6, we give a short survey on related work in section 5.

[1] see http://www.w3.org/DesignIssues/Principles.html

Semantic Web Workshop 2002 Hawaii, UAS

2. TECHNOLOGIES

This section provides an elaborated overview and evaluation of technologies providing modularization technology for the Web in general. Our overview is roughly separated into referring technologies and modularization technologies defined for XML.

2.1 Referring technologies

Links between Web resources are commonly called arches. Using or following a link for any purpose is called traversal. The traversal's origin is called the starting resource and the destination is the ending resource.

HTML Arches. present the simplest and oldest mechanism for referring in the Web. They provide simple outbound links between different resources which can be named for display purposes. Links to points inside other HTML documents are implemented using URI fragments.

XLink. [3] presents the referring technology adopted for XML and extends HTML's possibilities tremendously. Xlink allows to specify binary relations as found in RDF (see figure 1).

Additionally, it allows to link sets of elements with another by using arcs. Each set is qualified using a string label, all set elements use xlink:label with the same attribute value to specify set membership. In figure 2 for example, a one-to-many link has been generated, something neither possible in HTML nor in RDF. Interesting is also that XLink permits both inbound and outbound links. An inbound link is constituted by an arc from an external resource, located with a locator-type element, into an internal resource and is one possibility for modularity in XML.

Arc-type elements may have traversal attributes, one category are behavioral attributes which allows a certain kind of modularity, namely presentation time inclusion. If the attribute *show="embed"*

```
<!-- A local resource -->
      <actress xlink:label="maria">
          <firstname>Brigitte</firstname<
          <surname>Helm</surname<
     </actress>
<!-- A remote resource -->
      <movie xlink:label="metropolis"
             xlink:href="metropolis.xml"/>
<!-- An arc that binds them -->
      <acted xlink:type="arc" xlink:from="maria"
             xlink:to="metropolis"/>
```

Figure 1: Binary links with XLink

```
<divas xlink:title="German divas 1920s">
  <actress xlink:label="maria">
     <firstname>Brigitte</firstname>
     <surname>Helm</surname>
  </actress>

  <movie xlink:label="silent" xlink:title="Metropolis"
               xlink:href="metropolis.xml"/>
  <movie xlink:label="silent" xlink:title="Alaraune"
               xlink:href="alaraune.xml"/>
  <acted xlink:type="arc" xlink:from="maria"
                          xlink:to="silent"/>
        ...
<divas>
```

Figure 2: N-ary links for XLink

is stated for an Xlink arc, the referenced resource is embedded into the current document at interpretation-time (this is kínd of a lazy-load). The additional attribute *actuate* controls the event when the arc should be traversed[2].

This kind of lazy evaluation is problematic with regard to ontologies. As ontologies provide logical theories, "all knowledge" of any inferencing or deduction task must be gathered a-priori to ensure logical correctness. Additionally, the handling of an XLink is left to the application[3]

2.2 Inclusion technologies

In this subsection we focus on the modularization technologies recently proposed for XML. These technologies may be distinguished into:

1. External parsed (or text) entities, as defined by the XML 1.0 Recommendation [2]
2. XLinks (with embed behavior), as defined by the W3C XLink Working Draft.
3. XInclusions [10], as defined by the W3C XInclusion Working Draft.

External entities. An external parsed entity is declared in an XML (or SGML) document by an entity declaration without a notation. A reference to a parsed entity may occur practically anywhere in a document, between elements or within them, using the syntax &entity;. The entity itself may contain text, complete elements, or a mixture of them. It may not contain any declarations. XML does require that entity content be well-formed XML. In other words, one cannot have an element start tag in the document whose end tag is in a referenced entity, or vice versa. This is necessary so that a document may be checked to be well-formed even if the entity references are not replaced. This technology is not suitable for the Semantic Web, as the declaration of external entities requires DTDs. Additionally, the DOCTYPE declaration requires that the document element must be named, which is a unnecessary requirement.

XInclude [10]. is a processing model and syntax for general purpose inclusion. Inclusion is accomplished by merging a number of XML Infosets into a single composite Infoset. This implies that such processing occurs ̈below ̈the application level and without its intervention. Thus, the XInclude processor is responsible for validating the result infoset. The merging of infosets possibly leads to ID/IDREF conflict resolution and namespace preservation issues, not addressed by the working draft. Furthermore, the possibility to use XPOINTER range references exists, which makes maintainability questionable again.

XML Schema. The need for inclusion was also recognized for XML Schema. Due to the non-existence of general solutions at the time of creation a proprietary solution was sought. XML Schema [11] provides means to export certain elements of the schema for public usage. Additionally, facilities for importing elements from other schemas and a mechanism to completely include a referenced schema exist. This inclusion is not made visible to agents consuming the composite schema. This raises severe digital rights problems as the original provider is not recognizable anymore. The idea of defining a set of exported elements, stemming from experience with programming languages and distributed database schema definition systems, does not make sense for the Semantic Web. This export set suggests that there is value in distinguishing the internal implementation of a module from the features or interfaces that it provides for reuse by others[4], which is surely not the intent for ontologies.

3. EXTENDING RDF

This section is separated into two main parts. First, we collect different requirements for enabling a modularized Semantic Web. Second, these different requirements serve as input for defining a language extension to RDF that enables a modularized Semantic Web that recognizes the means offered by existing technologies.

3.1 Requirements

3.1.1 Import mechanisms

RDF only supports binary named links between different resources. While this is sufficient for schemaless metadata it only presents the basic technologies for conceptual relations, viz. the means for references between concepts in ontologies, but no means exist to determine whether referenced entities are actually defined and conceptually valid structures.

For example (see figure 3) the property *hasFather* in the *people ontology* is supposed to be a sub property of *hasParent*, which was defined in the *animal ontology*. Neither can one assure that *hasParent* exists nor that it is a property[6].

[2] traversal can take place onload or onactivate

[3] "...embedding affects only the display of the relevant resources; it does not dictate permanent transformation of the starting resource" [3]. If for example the ending resource is XML, it is not parsed as if it was part of the starting resource. Thus, embedded functionality of XLink is aimed at display behavior and not at true inclusion.

[4] [11] : "For example, a schema defined to describe an automobile might intend that its definitions for 'automobile' and 'engine' be building blocks for use in other schemas, but that other constructs such as 'screw' or 'bolt' be reserved for internal use.".

[6] Thus, the people ontology cannot be validated to be correct RDF

Animal ontology (available at &animal; [5]):

```
<rdf:RDF
  xmlns="&animal;#"
  xmlns:rdf="&rdf;#"
  xmlns:s="&s;#">
<s:Class rdf:about="#Animal"/>
<s:Class rdf:about="#Male">
  <s:subClassOf rdf:resource="#Animal"/>
</s:Class>
<s:Class rdf:about="#Female">
  <s:subClassOf rdf:resource="#Animal"/>
</s:Class>
<s:Class rdf:about="#Human">
  <s:subClassOf rdf:resource="#Animal"/>
</s:Class>
<s:Class rdf:about="#Lion">
  <s:subClassOf rdf:resource="#Animal"/>
</s:Class>

<rdf:Property rdf:about="#hasParent">
  <s:domain rdf:resource="#Animal"/>
  <s:range rdf:resource="#Animal"/>
</rdf:Property>

<rdf:Description rdf:about="#Marjan">
  <rdf:type rdf:resource="#Lion" />
</rdf:Description> </rdf:RDF>
```

People ontology

```
<rdf:RDF
  xmlns="&people;#"
  xmlns:rdf="&rdf;#"
  xmlns:s="&s;#">
<s:Class rdf:about="#Man">
  <s:subClassOf rdf:resource="&animal;#Human"/>
  <s:subClassOf rdf:resource="&animal;#Male"/>
</daml:Class>

<s:Class rdf:about="#Woman">
  <s:subClassOf rdf:resource="&animal;#Human"/>
  <s:subClassOf rdf:resource="&animal;#Female"/>
</s:Class>

<rdf:Property rdf:about="#hasFather">
  <s:subPropertyOf
        rdf:resource="&animal;#hasParent"/>
  <s:range
        rdf:resource="&animal;#Male"/>
</rdf:Property> </rdf:RDF>
```

Figure 3: Two Example RDF ontologies

Thus, in order to enable conceptual references across RDF models in a Web-like manner, we need a means to import entities that are defined somewhere else to take RDF out of ontological opaqueness. Clearly such an import primitive must locate the point of import. The established URIs without fragments suffice for this task, of course. We do not consider URNs here as they are not widely used.

3.1.2 Inclusion mechanisms

Inclusion mechanisms are different from import mechanisms with respect to the extension: Here, the complete RDF model is included whereas only specific parts are included with respect to importing.

Inclusion allows the decomposition of ontologies into individual parts and should therefore be a requirement for the Semantic Web, as it minimizes the effort to construct new ontologies. First, the overall effort required for the engineering of ontologies can be split among many shoulders. Second, decomposition not only simplifies construction and maintenance of ontologies, but also facilitates that ontologies become logically cohesive. Therefore leading to loosely coupled modules that denote single abstractions - a requirement for reusability in other applications.

Schema using tools such as the validating RDF parser [12]

3.1.3 Digital rights

It is clear that when using modularization in the Semantic Web one has to provide means for copyright management. This is needed in order to know who provided which information and created which artefact.

3.2 RDF Language Extension

3.2.1 rdfm:include

We propose to extend the basic RDF vocabulary by a new property rdfm:include (cf. figure 4) for the inclusion of another RDF model into the calling RDF model. This primitive can be understood by RDF parsers and specialized processors (working after parse time).

```
<rdf:RDF
  xmlns="&rdfm;#"
  xmlns:rdf="&rdf;#"
  xmlns:s="&s;#">

<!-- Source tagging -->

<rdf:Property rdf:ID="source">
  <s:comment>
  Identifies the source URI of a statement
  </s:comment>
  <s:domain rdf:resource="#Statement" />
</rdf:Property>

<!-- Inclusion mechanism -->

<rdf:Property rdf:ID="include" />

<!-- Import mechanisms -->

<rdf:Property rdf:ID="importFrom" />

<rdf:Property rdf:ID="transitiveImportFrom">
  <s:subPropertyOf rdf:resource="#importFrom"/>
</rdf:Property>

<rdf:Property rdf:ID="schemaAwareImportFrom">
  <s:subPropertyOf rdf:resource="#transitiveImportFrom"/>
</rdf:Property>
</rdf:RDF>
```

Figure 4: Modular RDF: the extended RDF vocabulary [7]

This property could be used in any RDF model to include the statements declared in another RDF model. Thus, the following statement would include the content of the animal ontology (cf. figure 3) in another RDF model:

```
<rdf:Description rdf:about="">
  <rdfm:include rdf:resource="&animal;"/>
</rdf:Description>
```

To meet the aforementioned requirement of digital rights, all statements found in the included RDF file have to be identified with their source. This can be implemented by tagging statement with their respective source URIs. Tagging statements can only be achieved using reification. Thus, we need to augment the RDF vocabulary (cf. figure 4) with a new property rdf:source that can only be validly applied to Statements [8].

[8] The reader may note that multiple source tags can be defined if identical statements occur in different source files

Thus, for each statement in an included RDF model, new reified statements are added to the calling RDF model. Consider the following statement in the animal ontology (cf. figure 3) for example

```
<s:Class rdf:about="&animal;#Animal"/>
```

This statement would be added in reified and tagged form, using the following set of statements:

```
<rdf:Description>

<rdf:subject rdf:resource="&animal;#Animal" />

<rdf:predicate
rdf:resource="&rdf;#type"/>

<rdf:object rdf:resource=&s;#Class"/>

<rdf:type
rdf:resource="&rdf;#Statement"/>

<rdfm:source rdf:resource="&animal;"/>

</rdf:Description>
```

3.2.2 *rdfm:import*

Notably, rdfm:include includes all statements found in another RDF file. This does not meet the demands of an import mechanism, which has to work on a lower granularity. To meet this demand, the RDF vocabulary has to be augmented with several new primitives that provide:

- A means to import statements about a given resource
- A means to transitively import statements about a given resource
- A means to schema-aware import of statements

rdfm:importFrom. In the simplest case only all statements on a given resource should be imported into the calling RDF model. We introduce a new property rdfm:importFrom to achieve this (cf. figure 4). The subject of a statement using the property[9] specifies the resource which should be imported into the calling RDF model whereas the object of the statement specifies the source RDF model, where statements about the subject should be taken from. Imported statements are represented in reified form with additional source identification to meet the digital rights requirement.

For example, the following statement

```
<rdf:Desciption rdf:about="&animal;#Male">
    <rdfm:importFrom rdf:resource="&animal;" />
</rdf:Description>
```

would add all statements about the resource *Male* from the animal ontology to the calling RDF model. In our example only one statement is found about Male, namely that it is a class. Hence, only one statement would be added:

```
<rdf:Description>

<rdf:subject rdf:resource="&animal;#Male"/>

<rdf:predicate
rdf:resource="&rdf;#type" />

<rdf:object
rdf:resource="&s;#Class"/>

<rdf:type
rdf:resource="&rdf;#Statement"/>

<rdfm:source rdf:resource="&animal;"/> </rdf:Description>
```

[9] These are statements of the following form: (resource, rdfm:importFrom, source)

rdfm:transitiveImportFrom. In many applications the importFrom is not sufficient as we can only embed the first level of resource references using importFrom. Consider the following example:

```
<rdf:Desciption rdf:about="&animal;#hasParent">
    <rdfm:transitiveImportFrom
       rdf:resource="&animal;"/>
</rdf:Description>
```

Here, the importFrom operation would only add statements that have hasParent as a subject, viz. the information that hasParent is a property whose domain is a resource Animal and whose range is again Animal. No information about Animal, e.g. that it is a class, would be included by importFrom.

The operation transitiveImportFrom targets this issue by additionally importing all statements on referenced resources. Thus, for the given example the following set of statements would additionally be pasted into the calling RDF model[10]:

```
<rdf:Description>

<rdf:subject rdfm:resource="&animal;#Animal"/>

<rdf:predicate
rdf:resource="&rdf;#type"/>

<rdf:object
rdf:resource="&s;#Class"/>

<rdf:type
rdf:resource="&rdf;#Statement"/>

<rdfm:source rdf:resource="&animal;"/>

</rdf:Description>
```

rdfm:schemaAwareImportFrom. One can easily see that even transitiveImportFrom is not sufficient to take RDF out of ontological opaqueness. For example if one transitively imports all information on the lion Marjan[11], of course all information on Marjan as well as all super classes of Lion are imported into the calling ontology but not the properties that are valid for any of these classes, thus another primitive is required to take the semantics of a RDF schema into account. Eventually

```
<rdf:Desciption rdf:about="&animal;#Marjan">
    <rdfm:schemaAwareImportFrom
    rdf:resource="&animal;" />
</rdf:Description>
```

would therefore add all statements on the property hasFather. Of course, the implementation of schemaAwareImportFrom is the most complex operation a processor has to fulfill.

Discussion. The following points are finally important to mention with respect to our presented proposal for extending RDF with import facilities:

[10] Additionally to the statements about *hasParent*, which are not shown for sake of brevity.

[11] Marjan is the lion who survived years of conflict and ill-treatment in Afghanistan and died at Kabul zoo. The 25 year-old beast who was half-blind, lame and almost toothless died of old age only weeks after an international animal rescue mission arrived to help him. The only lion in Kabul zoo, he was a gift from Germany in more peaceful times 23 years ago, and became something of a symbol of survival against the odds. Among his reported exploits are killing and eating a Taleban fighter who climbed into his enclosure to prove his bravery. The man's brother attacked the lion with a grenade in revenge, leaving it lame and blind in one eye.

- It does not rely on XInclude, as this approach is not applicable for RDF. The proposed merging of infosets would lead to two <RDF> </RDF> elements, which is incompatible with the RDF syntax specification. Furthermore, the source identification is not possible, as inclusion is invisible to the document consumers.

- The operation rdfm:include and the rdfm:importFrom property family are left in the RDF model. As statements from modules are only inserted at runtime this is not problematic. It is also important information for consuming agents that are interested in the way the viewed information is assembled.

- Modularization operations are generally transitive. Cyclic references are allowed. The usefulness of cyclic references has been shown in [4]. Even if cyclic references suggest that modules should be merged. We still keep the ability for organizational purposes. Infinite recursion can be avoided using simple means.

4. APPLICATIONS AND GUIDELINES

In this section we introduce applications where the foundations of our conceptual framework for modularizing RDF-based models have been successfully used. Additionally, we provide engineering guidelines for modularized ontologies.

4.1 Re-engineering Existing Resources

Experiences have shown that when developing an ontology-based system, conceptual resources, e.g. in the form of thesauri, lexical-semantic nets, related domain and application ontologies are already available. Furthermore, it has been seen that for the development of ontology-based information systems typically only parts of existing resources are to be used. Therefore, in our approach, we convert existing resources onto a common representation format, namely RDF-Schema. Based on this representation we generate application specific modules by applying bottom-up ontology pruning techniques based on a given set of text relevant for a specific domain [7].

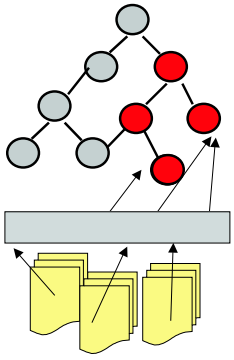

Figure 5: Applying Ontology Pruning to create modules

We take the assumption that the occurrence of specific concepts and conceptual relations in web documents are vital for the decision whether or not a given concept or relation should remain in an ontology. We take a frequency based approach determining concept frequencies in a corpus. Entities that are frequent in a given corpus are considered as a constituent of a given domain. To determine domain relevance ontological entities retrieved from a domain corpus are compared to frequencies obtained from a generic corpus. The user can select several relevance measures for frequency computation. The ontology pruning algorithm uses the computed frequencies to determine the relative relevancy of each concept contained in the ontology. All existing concepts and relations which are more frequent in the domain-specific corpus remain in the ontology. The user may also control the pruning of concepts that are neither contained in the domain-specific nor in the generic corpus. This pruning approach has been successfully applied in the following domains:

- GermaNet pruning for an insurance intranet application [7]: In this approach we used the German version of WordNet as a basis for generating an insurance-specific module, that supported an intranet-based knowledge management application.

- WordNet pruning for Reuters news document clustering: WordNet has been used as a semantic backbone for clustering Reuters news documents. The overall Wordnet lexical semantic net has been pruned on the basis of a set of selected Reuters documents, thus a news-specific module has been generated

- AGROVOC[12] pruning for the animal feed application: Finally, AGROVOC is a thesaurus provided by United Nations Food and Agricultural Organization, describing terms in the context of food and agriculture. It has been used as a basis for developing a module that exclusively describes the "animal feed" domain, for which a metadata-driven search engine will be built.

Using modularization techniques on top of the pruning results has the advantage that we do not create new ontologies. Instead we focus on the application specific part of a given ontology without defining new resources.

In general, we want to mention that there is a lack of modularity in current ontologies. Although, there exists the clear and good separation of top-level, domain, task and application ontologies (see [6]), there are no real-world ontologies and applications that are based on this principle. Most ontologies are not modular, neither by task, nor by domain. Therefore, ontology integration should modularize the namespace of a domain and separate task-oriented knowledge from the domain knowledge.

4.2 Engineering Guidelines

Much work has been described in the area of merging, mapping and integrating ontologies using very different approaches. The point of all those approaches is that they try to establish interoperability between syntactically and semantically heterogeneous conceptual models. Typically, this is done in an "ex-post" way, when the systems have already been established and are running.

Our engineering approach for modularized ontologies pursues another idea, namely the "ex-ante" establishment of interoperability. This approach allows the ontology engineer to import reusable modules from existing ontologies. The reader may note that this approach is already implicitly used in several existing commercial software products, e.g. in the area of knowledge management. Typically, these systems are divided in a standard basic conceptual model (describing basic concepts like documents, person's metadata, etc.) and a domain specific part of the conceptual model (describing domain-specific concepts like topic hierarchies, etc.). The point is that every KM application based on the basic conceptual data model can exchange data on this level, but it cannot exchange

[12] http://www.fao.org/agrovoc/

data on the domain-specific part of the conceptual model. We pick up this approach in an explicit way in the sense that we provide basic conceptual models in the form of ontology modules, e.g. for documents, persons, etc. There are many design goals within this modularization framework, e.g. to create coherent sets of semantically related modules, to support the creation of subsets and supersets of ontology modules for specific purposes, to facilitate future development by allowing modules to be upgraded or replaced independently of other modules and to encourage and facilitate the reuse of common modules by developers.

In the Semantic Web, modules have to be made accessible via a search engine for ontology modules. The search allows to query on a lexical (e.g. by providing a set of concept and property labels that should be contained in the module) and a conceptual layer (e.g. by providing a set of RDF statements that should be contained in the module). We are currently developing such a "ontology search engine" on the basis of our previous work for measuring the similarity between ontologies and parts of it [9].

5. RELATED WORK

Modularization is an established principle in software engineering, nevertheless, generally only acyclic inclusions are possible. Import mechanisms exist in all major programming languages and allow programmers to use classes, functions and methods defined in other modules by explicit naming.

Knowledge based systems introduced means for modularization in the early nineties. The LOOM system [8] provided an acyclic graph of inclusion relationships. Means for importing are not included, although references to symbols defined in other (non-included) ontologies are possible. Notably, the declarative semantics of ontologies are endangered by this, as the definition of those symbols is not visible.

Ontolingua [4] allows for modular organization in the ontology library system, organizes units into modules and allows cyclic inclusions. Additionally referenced ontologies can be extended by polymorphic refinement and restriction. RDF automatically supports some of those refinements (i.e. adding new domains and ranges[13]). Notably, we cannot support restrictions as RDF is not expressive enough to support the required translation axioms.

ONIONS [5] is a methodology that highlights the stratified design of ontologies. They propose different naming policies to achieve the modular organization or stratified storage of ontologies [5]. They show that disjointed partitioning of classes can facilitate modularity, assembling and integrating of ontologies.

As reported in section 2, several means for modularization are proposed for the XML world. XLink allows presentation time inclusion. Handling of inclusion in XLink is left to the application, which is not sufficient for the Semantic Web. No means for importing exist. XInclude introduces real inclusion for XML but does not allow for importing either. Furthermore the issue of conflicting XML identifiers is not recognized, which will come up in implementations.

XML Schema introduces proprietary means for inclusion as well as importing, but these operations are not made visible to agents consuming the composite schema. This raises severe digital rights problems as the original provider is not recognizable anymore. The idea of defining a set of exported elements, stemming from experience with programming languages and similar schema definition systems, does not make sense for the Semantic Web. This export set suggests that there is value in distinguishing the internal implementation of a module from the features or interfaces that it provides for reuse by others.

The recently proposed DAML+OIL ontology language recognized the need for modularity but only considers inclusion. Unfortunately, the established wording was not conceived. DAML+OIL follows the tradition of SHOE, where usage of other ontologies can be specified. Both approaches make the source of modular information opaque and present specialized approaches for ontologies only. Neither one presents means for imports.

6. CONCLUSION

Based on the general principle of modularization that has been an important design issue for the World Wide Web, and, in general the basis for powerful software systems, we have presented general principles and an approach for establishing a modularized Semantic Web.

Thus, ontologies and other RDF models can become a unit of composition which make context dependencies explicit. RDF models can be deployed independently and are subject to composition by third parties. This way, ontologies become logically cohesive, loosely coupled modules that denote single abstractions. Therefore they simplify the construction and maintenance and ensure reusability in other contexts. They also meet modularization requirement that was recognized for the upcoming Ontology Web Language[14].

In the future we plan to provide a simple implementation for a modularity processor and embed modularity management in our ontology engineering environment. We will also further our considerations of using URNs to allow replication of mission-critical RDF models.

7. REFERENCES

[1] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 2001.

[2] T. Bray, J. Paoli, and C.M. Sperberg-McQueen. Extensible markup language (XML) 1.0. Technical report, W3C, 1998. http://www.w3.org/TR/1998/REC-xml-19980210.

[3] Steve DeRose, Eve Maler, and David Orchard. XML Linking Language (XLink) Version 1.0, W3C Recommendation, 2001. http://www.w3.org/TR/xlink/.

[4] R. Fikes, A. Farquhar, and J. Rice. Tools for assembling modular ontologies in ontolingua. Technical Report KSL-97-03, Knowledge Systems Laboratory,Stanford University, 1997.

[5] Aldo Gangemi, Domenico M. Pisanelli, and Geri Steve. An overview of the ONIONS project: Applying ontologies to the integration of medical terminologies. *Data Knowledge Engineering*, 31(2):183–220, 1999.

[6] N. Guarino. Formal ontology and information systems. In *Proceedings of FOIS'98 – Formal Ontology in Information Systems, Trento, Italy, 6-8 June 1998*. IOS Press, 1998.

[7] J.-U. Kietz, R. Volz, and A. Maedche. A method for semi-automatic ontology acquisition from a corporate intranet. In *EKAW-2000 Workshop "Ontologies and Text", Juan-Les-Pins, France, October 2000.*, 2000.

[8] R. MacGregor. LOOM users manual. Technical Report ISI/EP-22, USC/ Information Sciences Institute, 1990.

[9] A. Maedche and S. Staab. Measuring similarity between ontologies. In *Technical Report, E0448, University of Karlsruhe*, 2001.

[13] Multiple ranges are demanded by the RDF Working Group

[14] See requirement 3 in the OWL requirements document, currently at http://km.aifb.uni-karlsruhe.de/owl

[10] Jonathan Marsh and David Orchard. XML Inclusions (XInclude) Version 1.0, W3C Working Draft , 2001. http://www.w3.org/TR/xinclude/.

[11] Henry S. Thompson, David Beech, Murray Maloney, and Noah Mendelsohn. XML Schema – W3C Recommendation, 2001. http://www.w3.org/TR/xmlschema-1/.

[12] Karsten Tolle. Validating rdf parser: A tool for parsing and validating rdf metadata and schemas. Master's thesis, University of Hannover, 2000.