D1.3.8/D2.5.7 User Syntaxes for Ontology Languages

Mustafa Jarrar
Rob Shearer
Boris Motik
Ian Horrocks
Christine Golbreich
Matthew Horridge

Abstract.
EU-IST Network of Excellence (NoE) IST-2004-507482 KWEB Deliverable D1.3.8 (WP1.3) / D2.5.7 (WP2.5)

Keyword list: ontology language, RDF, OWL, syntax, ORM, OBO

The RDF/XML serialization for OWL has proven quite successful as an interchange format, but it was never intended to be read or written by humans, nor to serve as an API for ontology manipulation—complex and verbose encodings make the encoding of even simple ontological structures quite arcane. In this deliverable we describe three different surface syntaxes which can be mapped to OWL: the Open Biological Ontologies format, a new Structured Ontology Format designed specifically to make working with OWL ontologies easier, and Object Role Modeling.

Document Identifier	KWEB/2007/D2.5.7/v1.0
Project	KWEB EU-IST-2004-507482
Version	1.0
Date	31 December, 2007
State	final
Distribution	public

Knowledge Web Consortium

This document is part of a research project funded by the IST Programme of the Commission of the European Communities as project number IST-2004-507482.

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

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

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

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

Free University of Bozen-Bolzano (FUB)
Piazza Domenicani 3
39100 Bolzano
Italy
Contact person: Enrico Franconi
E-mail address: franconi@inf.unibz.it

Institut National de Recherche en Informatique et en Automatique (INRIA)
ZIRST - 655 avenue de l'Europe -
Montbonnot Saint Martin
38334 Saint-Ismier
France
Contact person: Jérôme Euzenat
E-mail address: Jerome.Euzenat@inrialpes.fr

Centre for Research and Technology Hellas / Informatics and Telematics Institute (ITI-CERTH)
1st km Thermi - Panorama road
57001 Thermi-Thessaloniki
Greece. Po Box 361
Contact person: Michael G. Strintzis
E-mail address: strintzi@iti.gr

Learning Lab Lower Saxony (L3S)
Expo Plaza 1
30539 Hannover
Germany
Contact person: Wolfgang Nejdl
E-mail address: nejdl@learninglab.de

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

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

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

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

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

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

University of Sheffield (USFD)
Regent Court, 211 Portobello street
S14DP Sheffield
United Kingdom
Contact person: Hamish Cunningham
E-mail address: hamish@dcs.shef.ac.uk

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

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

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

Work package participants

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

Centre for Research and Technology Hellas /Informatics and Telematics Institute
Free University of Bozen-Bolzano
Institut National de Recherche en Informatique et en Automatique
University of Manchester
University of Trento
Vrije Universiteit Amsterdam
University of Innsbruck

Changes

Version	Date	Author(s)	Changes
0.0	21.05.07	Mustafa Jarrar	Creation
0.1	21.05.07	Mustafa Jarrar	Chapter 3 Included
0.2	28.05.07	Mustafa Jarrar	Chapter 3 revised according to Rob's comments
0.9		Rob Shearer	Compiled and edited to first draft
1.0		Rob Shearer	Edited in response to reviewer comments

Executive Summary

The OWL recommendation includes a well-defined text-based syntax for encoding ontologies based on the Resource Description Format (RDF). This syntax has proven quite successful as an interchange format, but it was never intended to be read or written by humans—complex and verbose encodings make the serialization of even simple ontological structures quite arcane. A more accessible surface syntax is needed.

Furthermore, there are a number of existing modeling languages which significantly predate OWL, and which enjoy wide use within specific user communities. By defining a mapping between these languages and OWL we both formalize their semantics and provide interoperability between the tools and ontologies available for these other languages, and the tools available for working with OWL.

In this deliverable we describe three different surface syntaxes which can be mapped to OWL: the Open Biological Ontologies format, a new Structured Ontology Format designed specifically to make working with OWL ontologies easier, and Object Role Modeling.

Contents

Chapter 1

Introduction

The Web Ontology Language (OWL) [BvHH$^+$04] has been defined by the World Wide Web Constortium (W3C) as part of its Semantic Web activity. Three dialects of the language have been defined, two of which are based on Description Logics—a well-understood family of knowledge representation formalisms with desirable computational properties. Formal semantics and the availability of efficient and provably correct reasoning tools have made the OWL DL dialect of OWL the language of choice for ontology development in fields as diverse as biology [SDCS05], medicine [GZB06], geography [Goo05], astronomy [DRPM06], geology,[1] agriculture [SLL$^+$04], and defense [LAF$^+$05]. Furthermore, OWL has been used to develop several large biomedical ontologies, such as the Biological Pathways Exchange (BioPAX) ontology [RRL05], the GALEN ontology [RR06], the Foundational Model of Anatomy (FMA) [GZB06], and the National Cancer Institute thesaurus [HdCD$^+$05]. Recently, the community of OWL users and developers proposed an extension of OWL called OWL 1.1 [PSH06a], which has been accepted as a member submission by the W3C. At the same time, a number of OWL-based tools have been developed, such as the Protégé [KFNM04a] and SWOOP [KPH05a] editors, and the FaCT++ [TH06], RACER [HM01], and Pellet [SP04] reasoners.

The OWL recommendation includes a well-defined text-based syntax for encoding ontologies based on the Resource Description Format (RDF). This syntax has proven quite successful as an interchange format, but it was never intended to be read or written by humans—complex and verbose encodings make the serialization of even simple ontological structures quite arcane. A more accessible surface syntax is needed.

Furthermore, there are a number of existing modeling languages which significantly predate OWL, and which enjoy wide use within specific user communities. By defining a mapping between these languages and OWL we both formalize their semantics and provide interoperability between the tools and ontologies available for these other languages, and the tools available for working with OWL.

[1] `http://sweet.jpl.nasa.gov/ontology/`

In this deliverable we describe three different surface syntaxes which can be mapped to OWL.

In Chapter 2 we describe OBO, an ontology language that has often been used for modeling ontologies in the life sciences. Its traditional definition is relatively informal, so we provide a clear specification for OBO syntax and semantics via a mapping to OWL. This mapping also allows us to apply existing Semantic Web tools and techniques to OBO. We show that Semantic Web reasoners can be used to efficiently reason with OBO ontologies. Furthermore, we show that grounding the OBO language in formal semantics is useful for the ontology development process: using an OWL reasoner, we detected a likely modeling error in one OBO ontology.

Chapter 3 presents a simple data model for the representation of OWL ontologies (including the new features of OWL 1.1). The model is built from basic structures native to all common programming environments, so it can be used directly as an API for ontology analysis and manipulation. Furthermore, serialization of these structures using the widely-supported YAML standard yields a readable text format suitable for ontology authoring by average users with text editors and code-management tools.

Finally, Chapter 4 focuses on Object Role Modeling (ORM), a graphical modeling language suitable for use by domain experts unfamiliar with logic or formal knowledge representation languages. ORM has long been used to encode the semantics of database systems. Our mapping describes a translation ORM to the $\mathcal{DLR}$ and $\mathcal{SHOIN}$ Description Logics, the latter of which can be translated to OWL, allowing the use of OWL reasoning engines to verify ORM models. A prototype implementation is described.

Chapter 2

Open Biomedical Ontologies Format

The Open Biomedical Ontologies (OBO) repository is a large library of ontologies from the biomedical domain hosted by the National Center for Biomedical Ontology (NCBO).[1] The majority of the ontologies in that repository are written in OBO Flat File Format[2]—an ontology language originally designed for the Gene Ontology (GO) [ABB$^+$00]. This language (from now on called simply OBO) uses a simple textual syntax that was designed to be compact, readable by humans, and easy to parse. The OBO community has dedicated significant effort to developing tools such as OBO-Edit[3]—an integrated OBO editor and reasoner.

In Section 2.1, we argue that there are many benefits in applying the tools and techniques of the Semantic Web to OBO. For example, Semantic Web reasoners could be used to provide guidance during ontology development; furthermore, modularization techniques for OWL [GHKS07] could simplify the reuse of existing OBO ontologies. This has been difficult up to now, however, since the OBO and Semantic Web communities have been largely disjoint.

To enable interoperability between OBO and Semantic Web tools and systems, we establish in Section 2.2 an exact relationship between OBO and OWL. This has not been straightforward, mainly because the OBO specification is quite informal. The syntax of the OBO language has not been formally specified, so our first step was to formalize the syntax of OBO itself; we discuss the results in Section 2.2.1. Likewise, there is no formal specification of OBO's semantics: the effects of different constructs have been described using natural language. We resolved ambiguities in these descriptions through extensive discussions with OBO developers. Hence, our mapping, presented in Section 2.2.2, formalizes the consensus interpretation in the OBO community. We also relate our mapping to several existing mappings from OBO to OWL.

In Section 2.3, we discuss how our mapping is used in practice. In Section 2.3.1 we

[1]`http://www.bioontology.org/repositories.html`
[2]`http://www.geneontology.org/GO.format.obo-1_2.shtml`
[3]`http://oboedit.org/`

discuss the technical aspects of our implementation. The complete replacement of OBO with OWL is not desirable for the OBO community, as many OBO users are familiar with both OBO-Edit and the OBO language, and find them convenient to use. Therefore, instead of simply implementing a translator from OBO to OWL, we have embedded support for OBO into existing Semantic Web ontology management infrastructure. In particular, we extended the well-known OWL API [HBN07] with an OBO parser and serializer. All tools built on top of the OWL API can thus directly load, process, and save OBO ontologies. Moreover, tools such as OBO-Edit could use the new API to provide similar features, including direct access to OWL reasoners.

In Section 2.3.2, we show that reasoners implementing the formal semantics of OWL can derive subsumption inferences that are missed by OBO-Edit's reasoner. In fact, on one of the OBO ontologies, our reasoner derived a new inference that highlights a probable modeling error.

Classifying large biomedical ontologies requires optimized reasoners. In Section 2.3.3, we show that OWL-based tools can be used to efficiently reason with OBO ontologies. To this end, we classified a number of OBO ontologies using the FaCT++ [TH06] and Pellet [SP04] systems, as well as the novel hypertableau-based reasoner HermiT [MSH07].[4] The design of HermiT was motivated by an analysis of the structure of biomedical ontologies such as GALEN and NCI. Our results show that HermiT's improved handling of GALEN is applicable to OBO ontologies as well: on several ontologies containing complex cyclic definitions of terms, HermiT outperforms the other reasoners by orders of magnitude. Thus, our mapping allows the OBO community to benefit from current and future advances in reasoning technology while continuing to use their familiar ontology language and tools.

2.1 Why Map OBO to OWL 1.1?

2.1.1 OBO at a Glance

An OBO ontology is a collection of *stanzas*, each of which describes one element of the ontology. A stanza is introduced by a line containing a *stanza name* that identifies the type of element being described. The rest of the stanza consists of lines, each of which contains a *tag* followed by a colon, a *value*, and an optional comment introduced by "!".

The following is an example of an OBO stanza defining the *term* (the OBO equivalent of a class) `GO:0001555` with name `oocyte growth`. This term is a subclass of the term `GO:0016049`. (The comment tells us that `GO:0016049` is named `cell growth`.) Furthermore, `GO:0001555` has a `part_of` relationship to the term `GO:0048601` (which is named `oocyte morphogenesis`). Finally, `GO:0001555` is defined as an intersection of the term `GO:0040007` (`growth`) and of a relationship

[4] `http://www.cs.man.ac.uk/~bmotik/HermiT/`

`has_central_participant` to `CL:0000023` (`oocyte`).

```
[Term]
id: GO:0001555
name: oocyte growth
is_a: GO:0016049 ! cell growth
relationship: part_of GO:0048601 ! oocyte morphogenesis
intersection_of: GO:0040007 ! growth
intersection_of: has_central_participant CL:0000023 ! oocyte
```

The following stanza defines the *relationship type* (the OBO equivalent of a property) `propreo:is_described_by`. The terms `propreo:chemical_entity` and `__Description177` are used as the domain and range, respectively, of the relationship type being defined.

```
[Typedef]
id: propreo:is_described_by
domain: propreo:chemical_entity
range: __Description177
```

Finally, the following stanza defines the *instance* (the OBO equivalent of an individual) `propreo:water_molecule`. The instance is made a member of the term `propreo:inorganic_solvent_molecule` and has `propreo:CHEBI_15377` for the value of the relationship `propreo:is_described_by`.

```
[Instance]
id: propreo:water_molecule
instance_of: propreo:inorganic_solvent_molecule
property_value: propreo:is_described_by propreo:CHEBI_15377
```

2.1.2 Why Formalize OBO Syntax?

The line-oriented syntax of OBO makes parsing ontologies into stanzas and tag-value pairs straightforward. The tag values, however, usually have a structure that depends on the tag type. This structure is described in the OBO specification in natural language. For example, the structure of `intersection_of` tag values is described as follows:

> This tag indicates that this term represents the intersection of several other terms. The value is either a term id, or a relationship type id, a space, and a term id. [...]

This style of description is quite informal and it does not make the conceptual structure of the OBO language clear. For example, the above description does not provide any intuition behind the distinction between the two alternative structures of allowed values. Furthermore, the specification of the structure is conflated with low-level lexical issues, such as whitespace handling. As a result, neither aspect of the language is robustly addressed; for example, the treatment of escape characters is dependent on the structure of tag values. These issues make the implementation of an OBO parser quite difficult in practice.

2.1.3 Why Formalize OBO Semantics?

The semantics of OBO is also defined informally, by providing natural-language descriptions for different types of tag-value pairs. For example, the OBO specification defines the semantics of the `relationship` tag as follows:

> This tag describes a typed relationship between this term and another term. [...] The `necessary` modifier allows a relationship to be marked as "not necessarily true". [...]

Such a description is clearly ambiguous and informal. The notion of a relationship being "necessarily true" is completely undefined; in fact, the notion of a relationship has not been formalized either. Computational logic can be used to provide an unambiguous interpretation for such statements. For example, the `relation` tag from the stanza for the term `GO:0001555` from Section 2.1.1 can be interpreted in at least three different ways:

- *Existantial interpretation*: Each instance of the term `GO:0001555` must have at least one `part_of` relationship to an instance of the term `GO:0048601`. This reading corresponds to the DL axiom $\texttt{GO:0001555} \sqsubseteq \exists \texttt{part\_of}.\texttt{GO:0048601}$.

- *Universal interpretation*: Instances of `GO:0001555` can be connected through `part_of` relationships only to instances of `GO:0048601`. This reading corresponds to the DL axiom $\texttt{GO:0001555} \sqsubseteq \forall \texttt{part\_of}.\texttt{GO:0048601}$.

- *Constraint interpretation*: Instances of the term `GO:0001555` can be connected through `part_of` relationships; furthermore, the end-points of the relationship must be *known* to be instances of `GO:0048601`. Such a statement cannot be formalized in standard DLs; however, it can be expressed in various extensions of DLs [MHS07, DNR02].

As another example, consider the natural-language explanation of the semantics for the `intersection_of` tag:

> [...] For example:
>
> ```
> intersection_of:  GO:00001
> intersection_of:  part_of GO:00002
> ```
>
> This means that the term is a subclass of any term that is both a subclass of `GO:00001` and has a `part_of` relationship to `GO:00002`. [...]

Here, it is not clear whether the defined term is equivalent to or a subclass of the intersection of the other terms. The textual description has a "procedural" flavor: it says that the defined term should be inferred to be a subclass of other terms, so one might conclude that the subclass relationship is the proper reading. Our discussions with the OBO developers, however, revealed that the intended interpretation is equivalence. The `union_of` tag suffers from analogous problems.

The description of OBO-Edit's reasoner provides an insight into the intended semantics of OBO. The OBO-Edit User's Guide[5] defines the following three reasoning rules:

1. For each transitive relationship R (such as `is_a` or `part_of`), whenever the ontology contains $a \rightarrow R \rightarrow b$ and $b \rightarrow R \rightarrow c$, an implied relationship $a \rightarrow R \rightarrow c$ is added.

2. For each term a defined as an intersection of terms b_1 and b_2, implied relationships $a \rightarrow$ `is_a` $\rightarrow b_1$ and $a \rightarrow$ `is_a` $\rightarrow b_2$ are added.

3. For each term a defined as an intersection of terms b_1 and b_2, whenever some term c has relationships $c \rightarrow$ `is_a` $\rightarrow b_1$ and $c \rightarrow$ `is_a` $\rightarrow b_2$, an implied relationship $c \rightarrow$ `is_a` $\rightarrow a$ is added.

This definition is procedural, and it misses important inferences. Consider the following example:

```
[Term]                      [Term]
id:  A                      id:  B
relationship:  R B          is_a:  C
```

This simple OBO ontology says that `A` has an `R`-relationship to `B`, and that `B` is a subclass of `C`. Regardless of which of the three previously suggested interpretations for the `relationship` tag we choose, we should derive that `A` has an `R`-relationship to `C`; however, OBO-Edit's reasoning procedure does not derive that. Furthermore, the second and third inference rules clearly state that `intersection_of` is interpreted as equivalence, which may be in conflict with the natural-language description of the semantics.

To sum up, OBO suffers from problems very similar to those identified in semantic networks [Qui68]. The DL family of ontology languages was developed precisely to

[5] Available as part of the OBO-Edit distribution.

address such problems—that is, to unambiguously specify the semantic properties of all ontology constructs. A mapping of OBO into OWL lends itself as an obvious way of providing formal semantics to OBO, and it allows for the application of sound and complete reasoning algorithms.

2.1.4 Why Use OWL 1.1?

In OBO, it is possible to make a property reflexive and/or (anti-)symmetric, as well as to say that one property is "transitive over" another: if P_1 is transitive over P_2, then for any individuals x, y, and z, the relationships $x \rightarrow P_1 \rightarrow y$ and $y \rightarrow P_2 \rightarrow z$ imply the relationship $x \rightarrow P_1 \rightarrow z$. Such axioms cannot be expressed in OWL DL; however, they can be expressed in the 1.1 extension of OWL. Thus, by using OWL 1.1 as the target language, we can capture a larger subset of OBO.[6] Since OWL 1.1 is fully backwards compatible with OWL, OBO ontologies that do not use any of the additional features of OWL 1.1 are mapped into OWL DL ontologies.

2.1.5 Reusing Existing Tools

An obvious practical benefit of a mapping from OBO to OWL is that it allows OBO users to exploit the multitude of existing OWL tools and services, instead of reimplementing the same functionality from scratch.

The foundation of many Semantic Web tools is provided by various APIs that provide means for the programmatic manipulation of ontologies. The OWL API [HBN07] is a prominent example of such an API that is now very widely used. Recently, it has been completely reengineered and made compliant with the OWL 1.1 version of the language. Jena[7] is a similar API that is comparable in its functionality with the OWL API and also has a large user base.

The OWL API has been used as the core data model for several ontology editors. For example, Protégé [KFNM04a] is a well-known editor that can be used to edit OWL ontologies. Its newest incarnation, Protégé 4, supports all of OWL 1.1 and is based on the new OWL API. SWOOP [KPH05a] is another OWL editor that is based on the OWL API. These editors have been developed over years and are de facto standards for ontology editing. Furthermore, they are imbued with functionality that can provide guidance to the user during the ontology development and maintenance processes; for example, SWOOP supports ontology debugging [KPSH05] and modularization [GHKS07]. Finally, a number of plug-ins have been written for Protégé, implementing features such as UML-based ontology editing and graph-based ontology visualization.

[6]Our translation captures all of the OBO 1.2 specification except for cyclic properties (the semantics of which is not completely clear) and negative assertions about properties (e.g., the assertion that a property is *not* transitive).

[7]`http://jena.sourceforge.net/`

Several highly optimized, sound, and complete reasoners for OWL exist. Pellet 1.4 [SP04] is built around the OWL API and is tightly integrated into SWOOP, and RacerPro [HM01], FaCT++ [TH06], and KAON2 [MS06] can be used with ontology editors through the Description Logics Implementors Group (DIG) protocol.[8] These reasoners can be used to classify an ontology and detect inconsistencies in class definitions, which is valuable during ontology development. Furthermore, reasoners can be used for query answering, which is the core inference underpinning many applications of OWL and OBO.

Apart from leveraging existing results, our integration allows the OBO community to reap the benefits of the future Semantic Web research. Conversely, OBO can provide to the OWL community significant user input as well as a growing library of OWL ontologies.

2.2 Providing a Formal Specification for OBO

In this section, we present a formal specification of the syntax and semantics of OBO. Due to space limitations, we highlight in this paper only the salient points; the full specification is available online.[9]

2.2.1 Formalization of OBO Syntax

We have formalized the OBO syntax by defining a BNF grammar, which maintains backward compatibility with the original OBO specification. Our grammar has been specifically designed to provide a conceptual description of the structure of OBO ontologies. To this end, it includes nonterminal symbols that identify and group certain OBO structures. Additionally, the grammar can be used to generate OBO parsers using automated tools.

For example, consider the definition of the `intersection_of` tag in BNF:

intersection	:=	`intersection_of:` ***termOrRestr***
termOrRestr	:=	***term-id*** | ***restriction***
restriction	:=	***relationship-id term-id***

As explained in Section 2.1.2, the value of the `intersection_of` tag can be either a term, or a relationship type followed by a term. Our grammar introduces structure to such a "flat" definition as follows. We introduce a nonterminal ***term-id***, which denotes a "named" term (mimicking OWL's named classes), and a nonterminal ***restriction***, which denotes a "restricted term" (mimicking OWL's restriction classes). Then, we introduce the nonterminal ***termOrRestr*** (mimicking OWL's complex classes). Finally, we say that the value of the ***intersection*** tag is a ***termOrRestr*** (mimicking OWL's intersection classes that can contain arbitrary classes as conjuncts).

[8]`http://dl.kr.org/dig/`
[9]`http://www.cs.man.ac.uk/~horrocks/obo/`

2.2.2 Mapping OBO to OWL

Our conceptualization of OBO's underlying model (described in Section 2.2.1) is quite similar to that of OWL, so the mapping between the two is relatively straightforward. We map OBO terms to OWL classes, OBO relationship types to OWL properties, and OBO instances to OWL individuals. The `id` assigned to each of these elements is used in OBO to uniquely identify each term, relationship type, and instance; hence, we use the value of `id` as the URI of the corresponding OWL element. The value of the `name` tag provides a human-readable description of OBO ontology elements, so it is translated into an OWL label.

Unlike OBO, OWL requires a strict separation between *object properties* (that relate individuals to each other) and *data properties* (that associate individuals with data values). To map OBO to OWL, we must infer which kind of property is appropriate for each OBO relationship type. If the range of a relationship type `R` is specified to be an XML datatype, or if `R` is asserted to be a subtype of another relationship type with such a range, then we translate `R` as an OWL datatype property; otherwise, we translate `R` as an object property.

OBO constructs such as `is_a`, `disjoint_from`, `domain`, and `range` have obvious equivalents in OWL and are translated in the straightforward manner. As discussed in Section 2.1.3, the official specification of the OBO language allows for several interpretations of the `intersection_of`, `union_of`, and `restriction` tags; hence, our mapping into OWL must pick the appropriate one.

Our discussions with OBO developers, as well as a survey of existing OBO ontologies, revealed that the existential interpretation (see Section 2.1.3) captures the intention behind the `relationship` tag. Hence, we translate the statement `relationship: R B` in a stanza defining the term `A` to the following OWL axiom:

```
SubClassOf(A ObjectSomeValuesFrom(R B))
```

Similarly, we concluded that the `intersection_of` tags should be interpreted as equivalences between classes (see Section 2.1.3). Furthermore, values given for the `intersection_of` tag that consist of a relationship type and a term should be interpreted as existential constraints, just like `relationship` tags. Hence, we translate the statements `intersection_of: C` and `intersection_of: R D` in a stanza defining the term `A` to the following OWL axiom:

```
EquivalentClasses(A
    ObjectIntersectionOf(C ObjectSomeValuesFrom(R D)))
```

There was no consensus on the formal semantics of several OBO constructs, such as the `not_necessary` modifier to the `relationship` tag and the use of `relationship` tags in `Typedef` stanzas. In fact, these tags are likely to be deprecated in future releases of OBO, and are currently treated as annotations by our mapping.

```
EntityAnnotation(OWLClass(GO_0001555) Label("oocyte_growth"))
SubClassOf(GO_0001555 GO_0016049) SubClassOf(GO_0001555
ObjectSomeValuesFrom(part_of GO_0048601))
EquivalentClasses(GO_0001555 ObjectIntersectionOf(
    GO_0040007 ObjectSomeValuesFrom(
        has_central_participant CL_0000023)))

ObjectPropertyDomain(propreo_is_described_by chemical_entity)
ObjectPropertyRange(propreo_is_described_by Description177)

ClassAssertion(propreo_water_molecule inorganic_solvent_molecule)
ObjectPropertyAssertion(
    is_described_by propreo_water_molecule CHEBI_15377)
```

Figure 2.1: The OWL Interpretation of the Stanzas from Section 2.1.1

Figure 2.1 shows the translation of the stanzas from Section 2.1.1.

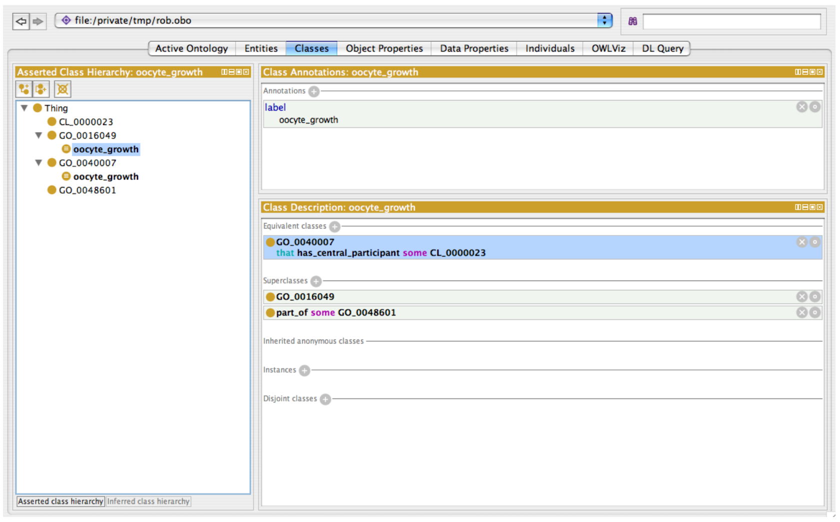
Related Work. Other mappings between the OBO Flat File Format and OWL exist, and a summary can be found online.[10] None of these mappings are based on a formal analysis of OBO. The Common Mapping—a new version of a mapping originally developed by Chris Mungall—is defined via an XSLT style sheet,[11] and has been implemented as a Protégé plug-in. Common Mapping differs from our mapping in several important respects. For example, it reifies OBO annotation values and turns them into first-class elements of the interpretation domain subject to consistency checking and inferencing; in contrast, our translation simply encodes them as OWL 1.1 annotations. Common Mapping translates neither `reflexive` nor `transitive_over` tags, whereas our encoding preserves their semantics. Finally, for several OBO ontologies, Common Mapping produces OWL Full ontologies, which cannot be efficiently processed by existing Semantic Web tools. In contrast, our translation produces OWL 1.1 ontologies, which can be processed by existing tools without problems.

2.3 Integrating OBO with the Semantic Web

2.3.1 Extending Semantic Web Tools to Support OBO

As mentioned, simply replacing the OBO language with OWL would not be desirable for the OBO community—OBO users are familiar with the existing syntax of the language

[10] `http://spreadsheets.google.com/ccc?key=pWN_4sBrd9l1Umn1LN8WuQQ`
[11] `http://www.godatabase.org/dev/xml/xsl/oboxml_to_owl.xsl`



```
▼ Thing
    CL_0000023
  ▼ GO_0016049
      oocyte_growth
  ▼ GO_0040007
      oocyte_growth
    GO_0048601
```

label
oocyte_growth

Equivalent classes:
GO_0040007
that has_central_participant some CL_0000023

Superclasses:
GO_0016049
part_of some GO_0048601

Figure 2.2: A Screenshot of the Example From Section 2.1.1 Viewed in Protégé 4

and the available tools, and want to continue to use them. Therefore, we adopted a less intrusive path of integration and have extended existing OWL tools with support for OBO.

As we discussed in Section 2.1.5, the OWL API lies at the core of many Semantic Web tools and supports fundamental tasks such as reading, saving, and manipulating ontologies. We extended the OWL API with an OBO parser and serializer, thus making OBO just another format supported by the API. This conveniently extends all applications based on the OWL API with support for OBO. For example, Protégé 4 can automatically read, edit, and save OBO ontologies; see Figure 2.2. Furthermore, the OWL API can be used to convert OBO files into OWL and vice versa by simply loading the file in one format and saving it in another. This functionality can be used to import OBO ontologies into tools that are not based on the OWL API and use custom OWL parsers.

The central new component in the API is an OBO parser, which consists of two distinct parts. The lower-level part is concerned with recognizing the syntax of OBO flat files, and it has been generated automatically from the BNF grammar described in Section 2.2.1. The upper-level part accepts different constructs from the OBO language and translates them into corresponding OWL 1.1 axioms according to the mapping described in Section 2.2.2.

2.3.2 Reasoning Support for OBO

An immediate benefit of our work is that it allows the application of Semantic Web reasoners to OBO ontologies. These reasoners are based on well-known algorithms with well-understood formal properties; furthermore, they provide formal guarantees about the completeness of reasoning, which makes the interpretation of derived results much easier. This can be quite useful in practice: on the OBO ontology `so.obo`, we used an OWL reasoner to detect a probable modeling error that is not detected by the OBO-Edit reasoner. This ontology contains the following stanzas that define the terms `SO:0000992` and `SO:0000914`:

```
[Term]                                  [Term]
id:  SO:0000992                         id:  SO:0000914
name:  BAC_cloned_genomic_insert        name:  cloned_genomic_insert
intersection_of:  SO:0000914
```

Note that the stanza for `SO:0000992` contains only one `intersection_of` tag. This seems to be a modeling error: presumably, the author simply forgot to add another `intersection_of` tag-value pair.

According to the mapping from Section 2.2.2, `intersection_of` defines a term to be equivalent to the intersection of other terms. Because the above intersection contains only one term, it effectively makes `SO:0000992` equivalent to `SO:0000914`. Indeed, OWL reasoners (correctly) derive that `SO:0000992` is a subclass of `SO:0000914` and vice versa. OBO-Edit's reasoner, however, only derives that `SO:0000992` is a subclass of `SO:0000914`, so the error remains undetected.

It is instructive to examine why OBO's reasoner does not derive the required inference. Namely, this inference could potentially be derived by applying the third inference rule from Section 2.1.3 on page 7 for $a =$ `SO:0000992` and $b_1 = b_2 = c =$ `SO:0000914`; however, for the rule to be applicable, we would need an `is_a` relationship from `SO:0000914` to itself. Semantically, each class is a subclass of itself; this fact, however, is not represented explicitly in the OBO ontology model, so the mentioned inference rule is not applicable.

This error might have been detected by checking whether each stanza contains at least two `intersection_of` tags. Since the syntax of the OBO language, however, has not been formally specified, it is hard to implement a comprehensive set of such checks, so errors often fall thorough to the semantic level. Furthermore, if our example stanza contained two `intersection_of` tags with the same value, the ontology would be syntactically correct, but would imply the same consequence. The OBO-Edit reasoner does not derive all inferences even with respect to the informal semantics, so such an error would not be detected at the semantic level either. In contrast, the syntax and the semantics of OWL 1.1 have been formally specified, which makes the detection of errors easier.

Table 2.1: Performance of Reasoning with OBO Ontologies

Tools	No. of ontologies classified in						
	200 ms	1 s	5 s	25 s	53 s	163 s	3925 s
Pellet	2	13	36	51	59	64	79
FaCT++	25	58	72	77	78	80	80
HermiT	48	65	74	81	82	82	82

2.3.3 Performance of Reasoning with OBO

Ontologies for the life sciences frequently contain many highly interconnected axioms with "cyclic definitions" of ontology terms. Such ontologies pose significant challenges to state-of-the-art tableau-based OWL reasoners [GHT06, MSH07]. Hence, it is interesting to see whether the OBO ontologies can be effectively handled using modern Semantic Web reasoning tools. Our mapping would clearly be much less useful if OWL reasoners were unable to process OBO ontologies.

Therefore, we conducted several reasoning experiments using different OBO ontologies and tools. In particular, we measured the times needed to compute the subsumption hierarchy of a large set of OBO ontologies. We used the well-known reasoners Pellet and FaCT++, and a new reasoner HermiT. The latter reasoner employs novel reasoning algorithms based on hypertableau and hyperresolution [MSH07]. The development of these algorithms has been motivated by an analysis of the structure of GALEN—the well-known biomedical terminology for which reasoning has proved to be quite hard. HermiT is currently the only reasoner that can classify the original version of GALEN [MSH07].[12]

Although the reasoning algorithms from [MSH07] support most of the OWL language, currently only the so-called Horn subset of OWL has been implemented in HermiT. Of the 88 ontologies available in the OBO repository, 83 fall into the supported fragment, so we used these ontologies in our performance tests. The ontologies are of varying sizes: the smallest one contains 166 axioms, whereas the largest one contains 37,943 axioms.

We summarize the times needed to classify these ontologies in Table 2.1. Because of the large number of ontologies, we do not present individual times for each ontology; instead, we just show how many ontologies each tool can classify within a certain time limit. The first four times were selected arbitrarily, whereas the last three times were chosen to show how long it takes for each tool to process the hardest ontology that it can classify.

[12]The original version of GALEN could not be processed by existing reasoners. Therefore, different versions of GALEN were derived from the original one by removing several cyclic definitions. Please refer to HermiT's web page for more information on this issue.

Our results show that HermiT efficiently deals with all but one ontology—that is, it classifies them in at most 53 seconds. FaCT++ exhausts the available resources on two ontologies that HermiT can classify. Thus, HermitT's novel reasoning techniques seem to be critical in dealing with complex ontologies. In the future, similar advances are likely to follow. By defining OBO in terms of a mapping to OWL, the OBO community can reap the benefits of these advances while continuing to enjoy the existing OBO syntax and tool set.

2.4 Conclusion

OBO is a language that has been extensively used for ontology modeling in the life sciences. Until now, the OBO language, as well as accompanying tools, have been developed independently from Semantic Web languages and tools. We argue that much can be gained from bringing the two languages closer together. On the one hand, this allows the OBO community to reuse OWL tools, while on the other hand, it provides new requirements and makes a large new corpus of ontologies available to the Semantic Web community.

The official specification of OBO is relatively informal. To obtain an unambiguous specification that can be easily implemented, we first formalized the syntax of the OBO language by capturing it in BNF. To capture the semantics, we developed a mapping between OBO and OWL. We have implemented this transformation in a new parser that we integrated into the OWL API, thus allowing numerous Semantic Web applications to use OBO as a native format. Finally, we showed that existing Semantic Web reasoners, such as HermiT, can be used to efficiently reason with OBO ontologies and can even identify likely modeling errors.

Our mapping provides a human-accessible ontology language and a bridge between the OBO and OWL communities, but OBO is still not an ideal language in all situations. Although the syntax is simpler than OWL's RDF/XML exchange syntax, OBO still requires the implementation of customer parsers. Further, the data model underlying OBO is not designed for programmatic access—there remains a large barrier to entry for tools which wish to manipulate complex OBO ontologies. Finally, not all constructs available in OWL can be easily expressed using OBO format. For this reason, users working with OWL ontologies are unlikely to benefit from use of an OBO syntax. Chapter 3 presents an alternative solution for these users.

Chapter 3

Structured Ontology Format

OWL standardization solves many of the interoperability problems which affected early DL systems; however, OWL's RDF/XML exchange syntax presents two challenges for developers:

1. The syntax is difficult for human users to read and write.
2. Parsing ontologies and working with the resulting ontology data (using some proprietary internal representation) require significant engineering effort.

The first issue has been addressed primarily through development of graphical tools for working with ontologies, such as Protégé [KFNM04b] and SWOOP [KPH05b]. Such tools make authoring accessible to inexperienced users, but graphical interfaces are forced to presume a particular user workflow and mindset which might not be appropriate in all cases; sophisticated editors include "expert mode" interfaces in which users directly manipulate text-based formats. Taking traditional programming as an analogy, graphical programming environments are helpful for those new to a programming language, and can even play a significant role in experienced engineers' workflows, but experienced developers rely on the ability to edit source code directly—a language without an accessible text-based format would be difficult to promote. As another parallel, it has frequently been noted that one of the primary advantages of HTML over its early competitors was the ability to easily examine and modify a web page's source code using a simple text editor. The complexity of RDF/XML makes text editing of OWL ontologies in that format extremely demanding, and graphical editors are not a sufficient replacement for a manageable syntax.

The second issue has led to the development of a number of sophisticated libraries which handle RDF parsing and allow programmatic access to ontology content. Systems such as Jena[1] offer direct access to an RDF model, while the WonderWeb OWL API

[1]`http://jena.sourceforge.net/`

[BVL03] and KAON2[2] include Java libraries with customized APIs for working with ontology structures at a higher level of abstraction than RDF graphs. Using such libraries does avoid the need for from-scratch parser implementation, but it also requires that developers learn new APIs and manage sometimes obscure library dependencies. More importantly, the primary advantage of OWL standardization has been lost: code for working with OWL ontologies is dependent not on the OWL standard, but on the particular API chosen for implementation. Most damningly, the current technology landscape suggests that working with OWL requires Java programming expertise. This puts OWL applications beyond the scripting skills of many biologists, and even out of reach of many web programmers who work mainly in Perl, Python, Ruby, and Javascript.

This paper addresses the stated problems with RDF/XML by offering a structured data model for ontologies built from primitive data types available in all major programming environments. Such a model is easily accessible to a wide range of implementors without the need for a proprietary API. Furthermore, the standard YAML [YAM] serialization of these structures provides a text ontology format appropriate for human authors. This ontology format has a standard translation to OWL 1.1 [PSH06b] (a superset of OWL DL) and is interpreted using OWL 1.1 semantics, and it supports all features of OWL 1.1 with the exception of datatypes and annotations. This paper is not meant to be a complete specification for all aspects of the format; an extended specification, conversion tools, tutorial code, and sample ontologies are available at the Structure Ontology Format web site.[3]

3.1 Background

Description logic notations derived from structured data models are not new. The KRSS syntax [PSS] was effectively a purely structural specification realized as "symbolic expressions" (S-expressions), the fundamental datatype in the LISP programming language based on linked lists of atoms (with a canonical serialization in LISP syntax). Such a "native" format was ideally suited to development of LISP reasoners and tools (and to ontology authoring by LISP programmers). Parsing and programmatic manipulation of S-expressions in other languages, however, is not widely supported, and while the syntax is very clean, the standard prefix notation and heavy use of parentheses feel unnatural to many users.

The W3C recognizes at least three different serializations for OWL: an *abstract syntax* [PSHH03] using a function-style format, the official exchange syntax [DCv$^+$02] based on RDF graphs (and serialized as XML encodings of these graphs), and a rarely-used pure XML syntax [HEPS03] defined in a "W3C Note". The Description Logic Implementors Group[4] has defined an alternative XML language for encoding description logic ontolo-

[2]`http://kaon2.semanticweb.org/`

[3]`http://www.cs.man.ac.uk/~rshearer/sof/`

[4]`http://dl.kr.org/dig/`

gies. (The next version of the DIG specification is expected to share a pure XML syntax with the OWL 1.1 proposal.)

XML does offer a formal data model, so XML-based formats can be viewed as structural specifications and processed with one of the many XML toolchains available. Such tools can be integrated with most programming environments, but "native" manipulation of XML structures (and appropriate mechanisms for abstraction of irrelevant low-level details) is available in only very specialized languages (such as XSLT and XQuery). Furthermore, XML syntax is not optimized for direct human interaction.

As described in Chapter 2, the Open Biomedical Ontologies[5] effort has independently developed a standardized encoding for ontologies[6], with human-readable syntax and simple parsing as major design goals. The OBO syntax breaks an ontology document into labeled sections called "stanzas"; a stanza contains a set of "tags", each specified in `key: value` format on a single line. Some stanzas define "terms" (comparable to OWL classes), and some describe "instances" (comparable to OWL individuals). OBO format is very accessible to human users, and our mapping from OBO to OWL formalizes OBO semantics, but the syntax requires a custom parser, there is no obvious programmatic API for OBO data, and the language does not provide the same formal expressiveness as OWL.

Graphical modeling tools frequently need to display complex class expressions to users, and this was initially done using formal logic symbols (e.g. $\exists R.(C \sqcap D)$). In order to make such descriptions more accessible to nonlogicians, the *Manchester OWL Syntax* (MOS) [HDG$^+$06] was designed to use infix notation and read as natural language. (The above expression would be written in MOS as "R some (C and D)".) A simple frame-based syntax was described for text exchange of class definitions, but the approach was not extended to a full ontology language. The structured format presented in this paper incorporates a formalized, extended version of Manchester syntax for complex class descriptions (see Section 3.2.2).

3.2 Structured Ontology Format

3.2.1 Data Model

The Structured Ontology Format (SOF) data model is based on three types of structure: character strings (the only atomic type) store text; *collections* directly contain other structures; and *maps* associate keys with values. The representational details of these types are unimportant: map and collection types may be ordered or unordered, duplicate values are allowed (but never required) in collections, and maps may contain a single binding or multiple bindings for the same key.

[5] `http://obo.sourceforge.net/`

[6] `http://www.godatabase.org/dev/doc/obo_format_spec.html`

The ontology format is dependent upon an *expression language* used to encode complex class descriptions, properties, and individual names. Expression languages are detailed in Section 3.2.2; for the purposes of this discussion, classes, properties, and individuals are assumed to be identified using strings.

An ontology is a map. If the string `classes` is present as a key in the ontology, then its binding is either a collection of class identifiers, or a map whose keys are class identifiers and whose values are *frames* describing the classes to which they are bound. Such frames are maps; if a frame bound to C binds the string `subsumed by` to a collection containing class identifier D, then D subsumes C is an axiom of the ontology. Bindings within class frames for keys such as `equivalent to` and `disjoint from` are interpreted analogously.

Other keys in an ontology have similar meanings: the value bound to the `properties` key is a collection or a map from property identifiers to frames describing those properties, and that bound to `individuals` is a collection or a map to individual frames.

The bindings for the `facts`, `class axioms`, and `property axioms` keys within an ontology are not maps but collections containing axioms not directly associated with any particular class, property, or individual. Within the `class axioms` collection, a map containing a single binding from `disjoint` to a collection of class identifiers asserts that all of the specified classes are disjoint from one another.

The formal semantics for SOF ontologies are given by a correspondence with OWL 1.1 functional syntax, presented in Table 3.1. We use a small subset of YAML notation to represent structures: [x_1,...,x_n] is a collection containing elements x_1 through x_n, and $\{x : y\}$ is a map in which the binding for key x is y. A path syntax identifies the bindings for keys within nested maps: "/foo" indicates the value associated with key `foo` in the ontology map, and "/foo/bar" identifies the binding for key `bar` within "/foo". An object x *contains* value y if x is a collection containing y as an element, or if x is a map with an assignment for key y. Finally, for expression x in a structured ontology, the term $\bar{x}$ within an OWL axiom represents the translation of x to a class, property, or individual expression in OWL functional syntax, in accordance with the chosen expression language (described in Section 3.2.2).

The OWL 1.1 ontology derived from an ontology in SOF includes the axiom in the third column of Table 3.1 for every value identified by the path in the first column which contains the structure given in the second column. Converting an ontology to SOF from OWL 1.1, however, can be done in a number of ways. Most OWL axioms can be encoded in several different ways in SOF; e.g. `SubClassOf(`c`,` d`)` can be encoded by adding c to "/classes/d/subsumes", adding d to "/classes/c/subsumed by", or adding "$\{c : d\}$" to "/class axioms". Any translation need choose only one such translation. Rows 1–33 are usually only applicable to restricted forms of the OWL 1.1 axioms (e.g. equality axioms involving only two elements); all OWL 1.1 axioms can be transformed to SOF using rows 34–56.

Table 3.1: OWL interpretation of Structured Ontology Format

	SOF structure	**Contains**	**OWL 1.1 Equivalent**
1	/classes	c	Declaration(OWLClass($\bar{c}$))
2	/classes/c/subsumed by	d	SubClassOf($\bar{c}$ $\bar{d}$)
3	/classes/c/subsumes	d	SubClassOf($\bar{d}$ $\bar{c}$)
4	/classes/c/equivalent to	d	EquivalentClasses($\bar{c}$ $\bar{d}$)
5	/classes/c/disjoint union of	$[d_1,...,d_n]$	DisjointUnion($\bar{c}$ $\bar{d_1}$ ... $\bar{d_n}$)
6	/classes/c/disjoint from	d	DisjointClasses($\bar{c}$ $\bar{d}$)
7	/classes/c/domain of	r	ObjectPropertyDomain($\bar{r}$ $\bar{c}$)
8	/classes/c/range of	r	ObjectPropertyRange($\bar{r}$ $\bar{c}$)
9	/classes/c/members	i	ClassAssertion($\bar{i}$ $\bar{c}$)
10	/properties	r	Declaration(ObjectProperty($\bar{r}$))
11	/properties/r/subsumed by	s	SubObjectPropertyOf($\bar{r}$ $\bar{s}$)
12	/properties/r/subsumes	s	SubObjectPropertyOf($\bar{s}$ $\bar{r}$)
13	/properties/r/subsumes	$[s_1,...,s_n]$	SubObjectPropertyOf(SubObjectPropertyChain($\bar{s_1}$ ... $\bar{s_n}$) $\bar{r}$)
14	/properties/r/equivalent to	s	EquivalentObjectProperties($\bar{r}$ $\bar{s}$)
15	/properties/r/inverse	s	InverseObjectProperties($\bar{r}$ $\bar{s}$)
16	/properties/r/disjoint from	s	DisjointObjectProperties($\bar{r}$ $\bar{s}$)
17	/properties/r/domain	c	ObjectPropertyDomain($\bar{r}$ $\bar{c}$)
18	/properties/r/range	c	ObjectPropertyRange($\bar{r}$ $\bar{c}$)
19	/properties/r	functional	FunctionalObjectProperty($\bar{r}$)
20	/properties/r	inverse functional	InverseFunctionalObjectProperty($\bar{r}$)
21	/properties/r	reflexive	ReflexiveObjectProperty($\bar{r}$)
22	/properties/r	irreflexive	IrreflexiveObjectProperty($\bar{r}$)
23	/properties/r	symmetric	SymmetricObjectProperty($\bar{r}$)
24	/properties/r	asymmetric	AntisymmetricObjectProperty($\bar{r}$)
25	/properties/r	transitive	TransitiveObjectProperty($\bar{r}$)
26	/properties/r/related	$\{i : j\}$	ObjectPropertyAssertion($\bar{r}$ $\bar{i}$ $\bar{j}$)
27	/properties/r/not related	$\{i : j\}$	NegativeObjectPropertyAssertion($\bar{r}$ $\bar{i}$ $\bar{j}$)
28	/individuals	i	Declaration(Individual($\bar{i}$))
29	/individuals/i/same as	j	SameIndividual($\bar{i}$ $\bar{j}$)
30	/individuals/i/different from	j	DifferentIndividuals($\bar{i}$ $\bar{j}$)
31	/individuals/i/member of	c	ClassAssertion($\bar{i}$ $\bar{c}$)
32	/individuals/i/related/r	j	ObjectPropertyAssertion($\bar{r}$ $\bar{i}$ $\bar{j}$)
33	/individuals/i/not related/r	j	NegativeObjectPropertyAssertion($\bar{r}$ $\bar{i}$ $\bar{j}$)
34	/facts	$\{i : c\}$	ClassAssertion($\bar{i}$ $\bar{c}$)
35	/facts	$\{\{i : j\} : r\}$	ObjectPropertyAssertion($\bar{r}$ $\bar{i}$ $\bar{j}$)
36	/facts	$\{$same : $[i_1,...,i_n]\}$	SameIndividual($\bar{i_1}$ ... $\bar{i_n}$)
37	/facts	$\{$different : $[i_1,...,i_n]\}$	DifferentIndividuals($\bar{i}$ $\bar{j}$)
38	/facts	$\{$not related : $\{\{i : j\} : r\}\}$	NegativeObjectPropertyAssertion($\bar{r}$ $\bar{i}$ $\bar{j}$)
39	/class axioms	$\{$disjoint : $[c_1,...,c_n]\}$	DisjointClasses($\bar{c_1}$ ... $\bar{c_n}$)
40	/class axioms	$\{$equal : $[c_1,...,c_n]\}$	EquivalentClasses($\bar{c_1}$ ... $\bar{c_n}$)
41	/class axioms	$\{c : d\}$	SubClassOf($\bar{c}$ $\bar{d}$)
42	/class axioms	$\{$disjoint union : $\{c : [d_1,...,d_n]\}\}$	DisjointUnion($\bar{c}$ $\bar{d_1}$ ... $\bar{d_n}$)
43	/property axioms	$\{$disjoint : $[r_1,...,r_n]\}$	DisjointObjectProperties($\bar{r_1}$ ... $\bar{r_n}$)
44	/property axioms	$\{$equal : $[r_1,...,r_n]\}$	EquivalentObjectProperties($\bar{r_1}$ ... $\bar{r_n}$)
45	/property axioms	$\{r : s\}$	SubObjectPropertyOf($\bar{r}$ $\bar{s}$)
46	/property axioms	$\{[r_1,...,r_n] : s\}$	SubObjectPropertyOf(SubObjectPropertyChain($\bar{r_1}$ ... $\bar{r_n}$) $\bar{s}$)
47	/property axioms	$\{$functional : $r\}$	FunctionalObjectProperty($\bar{r}$)
48	/property axioms	$\{$inverse functional : $r\}$	InverseFunctionalObjectProperty($\bar{r}$)
49	/property axioms	$\{$reflexive : $r\}$	ReflexiveObjectProperty($\bar{r}$)
50	/property axioms	$\{$irreflexive : $r\}$	IrreflexiveObjectProperty($\bar{r}$)
51	/property axioms	$\{$symmetric : $r\}$	SymmetricObjectProperty($\bar{r}$)
52	/property axioms	$\{$asymmetric : $r\}$	AnitsymmetricObjectProperty($\bar{r}$)
53	/property axioms	$\{$transitive : $r\}$	TransitiveObjectProperty($\bar{r}$)
54	/property axioms	$\{$domain : $\{r : c\}\}$	ObjectPropertyRange($\bar{r}$ $\bar{c}$)
55	/property axioms	$\{$range : $\{r : c\}\}$	ObjectPropertyDomain($\bar{r}$ $\bar{c}$)
56	/property axioms	$\{$inverse : $\{r : s\}\}$	InverseObjectProperties($\bar{r}$ $\bar{s}$)

3.2.2 Expression Language

The structures used to encode axioms in SOF are at most five levels deep, but OWL class descriptions (and property expressions) can be very complex, with arbitrarily deep nesting of subexpressions. It is difficult to define a single format for such a language which is both readable for humans and easy for machines to process.

The structured format described in Section 3.2.1 is independent of the language used for class descriptions, property expressions, and individual identifiers. This allows different dialects of structured format to use different expression languages.

In order to eliminate the need for text parsing, a structured expression language similar to KRSS but with OWL 1.1 expressiveness has been formalized. (Details are available on the SOF web site.[7]) For many users, however, a more lightweight, readable encoding closer to natural language is preferable. For this reason, a second dialect of structured ontology format is defined which encodes class, property, and individual expressions as simple (unicode) strings. These strings are interpreted as Extended Manchester OWL Syntax (EMOS), which we summarize here. A formal grammar and translation to OWL 1.1 are given at the Manchester OWL Syntax web site.[8]

EMOS is backwards-compatible with the original Manchester syntax presented in [HDG+06]. Manchester syntax was designed as a simple and readable text format for expressing complex class descriptions, intended primarily for presenting such descriptions to (non-logician) human users. The focus is on allowing even relatively complex expressions to be readable as natural (English) language: ObjectIntersectionOf, ObjectUnionOf, and ObjectComplementOf OWL descriptions are written with `and` (or `that`), `or`, and `not`, class expressions involving properties (such as ObjectSomeValuesFrom and ObjectHasValue) are expressed using infix notation (with keywords `some`, `only`, `min`, `max`, `exactly`, and `value`), and parentheses are needed only where rules of precedence demand them. The language also includes a number of shorthands for common modeling patterns: the `someonly` construct stands for the intersection of multiple `some` and `only` expressions, `never R` is short for `R max 0`, and `always R` means `R some owl:Thing`. A property `P`'s inverse is given by `P-`, and class, property, and individual names can be quoted to prevent conflicts with the language keywords.

Identifers in EMOS can be explicitly typed as full RFC 3987 IRIs (surrounded by "<" and ">" tokens), or they can be interpreted subject to namespace expansion similar to that described in [BHLT06]. Crucially, interpretation of EMOS expressions is dependent upon a set of *namespace bindings* mapping namespaces prefixes to their expansions. In Structured Ontology Format, these bindings are given by a map stored as the value of the `namespaces` key in the ontology object (with the default namespace bound to a null or empty prefix). Unlike in standard XML, a namespace prefix used in a legal EMOS identifier need not be declared; the interpretation of such identifiers is application-dependent.

[7]http://www.cs.man.ac.uk/~rshearer/sof/sel
[8]http://www.cs.man.ac.uk/~rshearer/mos/

Although this expression language is relatively straightforward to parse, such string processing is far more complex than manipulation of an SOF ontology's axiom structure. Ontology authors and authoring tools should be mindful of the fact that some SOF processors may view class descriptions, property expressions, and individual identifiers as opaque strings and might be unable to perform even namespace expansion. When the same expression occurs multiple times within an ontology, it is recommended that all encodings of that expression be lexically identical, particularly when the expression is a simple identifier.

3.2.3 Serialization

YAML[9] is a standard serialization format for simple data structures, including maps, ordered collections (called *sequences*), and string values. The syntax is extremely flexible, with a number of different styles of encoding for each data type. Sequences, for example, can be written as comma-separated lists of values within square brackets (as in Table 3.1), or elements can be listed on separate lines, with each element preceded by a dash. Nested sequences are distinguished by indentation level. Bindings within maps are written as `key : value`, and maps can be enclosed in curly braces with comma-separated bindings, or they can be written in a line-oriented style similar to sequences. The syntax also allows comments (introduced by `#`, and continuing to the end of the line). The full YAML specification is defined in [YAM]. A small example of a complete SOF ontology serialized as YAML is shown in Figure 3.1.

While YAML is the canonical (and most human-friendly) serialization format for SOF ontologies, in some cases more restrictive encodings are useful. The JSON[10] fragment of YAML allows only the braced/bracketed forms of maps and sequences (among other restrictions), and as a result JSON is extremely simple to parse. SOF ontologies using Structured Expression Language and serialized as JSON are ideally suited to efficient machine-to-machine exchange of OWL ontology data.

3.2.4 Special Forms

Most OWL axioms can be encoded in two or even three different ways in structured format—for example an ObjectPropertyDomain axiom in an OWL ontology can be represented in SOF according to rows 7, 17, or 54 of Table 3.1 (modulo class and property declarations). This provides a great deal of flexibility for ontology authors, but it is problematic for tools (or humans) scanning an ontology for a particular type of axiom. We thus define a number of *normal forms* which add requirements that certain axioms be encoded in particular ways.

[9]`http://yaml.org/`
[10]`http://www.json.org/`

For a given row r of Table 3.1, OWL ontology K, and SOF representation of K S, if every axiom of K which could be represented in structured format in accordance with row r is represented in that way in S, then we say that S is *normalized* with respect to r. Such normalization does not require reasoning about semantic entailments between axioms: normal forms deal only with different ways of encoding the same structural/syntactic content.

An ontology is said to be *class-frame normalized* if it is normalized with respect to rows 1–6 of Table 3.1. Note that class-frame normalization does not require the encoding of domain, range, or membership axioms within the class frame.

Analogously, *property-frame normalized* ontologies are normalized with respect to rows 10–25, and *individual-frame normalized* ontologies are normalized with respect to rows 28–33. A *fully frame-normalized* ontology is normalized with respect to rows 1–33 (and thus must be individual-frame, property-frame, and class-frame normalized), and an *axiom-normalized* ontology is normalized with respect to rows 34–56. A *fully normalized* ontology is normalized with respect to all rows of Table 3.1.

In addition to these simple syntactic conditions on an ontology's SOF representation, three criteria on the underlying ontology data are very useful for ontology authoring and processing tools. First, if all namespace prefixes used in all class, property, and individual expressions throughout the ontology are declared in the `namespaces` mapping, then the ontology is *namespace consistent*. Without this property, there is no guarantee that two different translations of the same ontology into OWL will entail each other. Second, an ontology with declarations for all class, property, and individual names is *structurally consistent*. (This notion is from the OWL 1.1 specification.) An ontology which is both namespace and structurally consistent can benefit from extensive authoring support (highlighting of typos, tab completion, etc.). Finally, given some definition of equivalence between class descriptions (lexical equality is sufficient), an ontology is *taxonomy optimized* if no class subsumption axioms are redundant with respect to the transitive-reflexive closure of all subsumption axioms in the ontology.

3.3 Discussion and Future Work

Structured format has already proven to be extremely useful as a language for creating test suites for new reasoner implementations: the syntax makes it possible to author simple tests (with dozens of axioms) by hand, and more complex tests are easy to construct and serialize using standard scripting languages. Furthermore, implementation of a parser for OWL's RDF/XML syntax can be substantially more expensive than construction of naïve reasoners for restricted logic fragments. The use of a Java tool based on one of the existing OWL parsing libraries to "preprocess" ontologies into structured format has allowed realistic knowledge bases to be used as tests for new reasoning algorithms prototyped in Perl and Python.

Conversion of ontologies to SOF and examination with text tools such as `grep` has replaced ontology exploration using graphical tools in some workflows. Version control of ontologies maintained in structured format has proven much easier than with other formats: text difference tools allow the same conflict-resolution strategies as are common with traditional programming languages.

Straightforward programmatic access to ontology data using modern dynamic languages has decreased the engineering cost for developing new OWL tools by several orders of magnitude. Routines to convert ontologies between the special forms described in Section 3.2.4 can be implemented in under a dozen lines of Python code (and only a few minutes' work), and a from-scratch ontology exploration interface intended for visually-impaired users weighs in at under one hundred lines (and roughly an hour of implementation time).

The current specification, however, offers only partial coverage of OWL 1.1: neither datatypes, datatype properties, nor annotations are currently supported. Further, there is no "import" functionality for structured ontologies. Networks of OWL imports are converted to SOF as a single monolithic text file, which can be difficult to manage. An extension of structured format which includes these features is currently being developed.

```
namespaces:
    "" : http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#
    food : http://www.w3.org/TR/2003/PR-owl-guide-20031209/food#

classes:
    Wine:
        subsumed by:
            - food:PotableLiquid   # note use of namespace
            - hasMaker exactly 1   # class descriptions use EMOS
            - locatedIn some Region
    TableWine:
        equivalent to:
            - Wine that hasSugar value Dry
properties:
    hasMaker:
        inverses: [ producesWine ] # bracketed syntax for sequences
        functional:                # some keys don't need values
    locatedIn:
        range: [ Region ]
        transitive:
individuals:
    StonleighSauvignonBlanc:
        member of:
            - Wine
        related:
            hasSugar: [ Dry ]      # sequences can be bracketed...
            hasMaker:
                - Stonleigh        # ...or use a line-oriented style.
            locatedIn: [ NewZealandRegion ]
```

Figure 3.1: A small portion of the Wine ontology in SOF, using Manchester syntax and YAML serialization

Chapter 4

Object Role Modeling

Object Role Modeling (ORM) is a conceptual modeling method that allows the semantics of a universe of discourse to be modeled at a highly conceptual level and in a graphical manner. ORM has been used commercially for more than 30 years as a database modeling methodology, and has recently becoming popular not only for ontology engineering but also as a graphical notation in other areas such as the modeling of business rules[Hal04, Nor99, DJM02, Jar06a], XML-Schemes[BGH99], data warehouses[Pip06], requirements engineering[HPH04, BB95], web forms[Jar05, DAHtH02], and web engineering[DT05].

We formalize the ORM using both the $\mathcal{DLR}$ and the $\mathcal{SHOIN}$/OWL Description Logics. This would enable automated reasoning to be carried out on the formal properties of ORM diagrams (and thus legacy database systems), such as detecting contradictions, implications, and inference. In addition, the expressive, methodological, and graphical power of ORM make it a good candidate for use as a graphical notation for most description logic languages. With this, non-IT trained industrial experts will be able to build axiomatized theories (such as ontologies, business rules, etc.) in a graphical manner, without having to know the underpinning logic or foundations.

Indeed, in many domains the ontology building process is difficult and time consuming. From practical experience, this is not because these domains are not well understood or that a consensus cannot be made about them. Typically, this is because it is difficult for domain experts to understand and use ontology languages. Current ontology languages (and tools) require an understanding of their underpinning logic. The limitation of these languages is not that they lack expressiveness or logical foundations. Instead, it is in their capability to be used by subject matter experts.

Certain properties are required for an ontology language to be easily understood by domain experts. First, its constructs should be close to the language that business experts speak and the "logic" they use. For example, it is easier for a lawyer to say "it is mandatory for each Complaint Problem to be testified by at least one Evidence", than to say "the cardinality between the concept Complaint Problem and the concept Evidence is (1:0)". Second, the language should have a graphical notation to enable simple conceptual modeling. By ”graphical notation” here, we refer to not only visualization, but also a “graphical language” that allow domain experts to construct an ontology using a graphical notation for each concept, relation, or rule type. In other words, such a language should guide experts through its modeling constructs to “think” conceptually while building, modifying, or validating an ontology.

4.0.1 Object Role Modeling (ORM)

Many commercial and academic tools that support ORM solutions are available, including the ORM solution within Microsoft’s Visio for Enterprise Architects [HEHM03], VisioModeler [Vis], NORMA [NOR], CaseTalk [Cas], Infagon [Inf], and DogmaModeler [Hal05]. DogmaModeler and its support for ontology engineering will be presented later.

ORM has an expressive and stable graphical notation. It supports not only n-ary relations and reification, but also a fairly comprehensive treatment of many practical and standard business rules and constraint types. These language features include identity, mandatoriness, uniqueness, subsumption, totality, exclusivity, subset, equality, exclusion, value, frequency, symmetry, intransitivity, acyclicity, derivation rules, and several others. Furthermore, compared with, for example, EER or UML, ORM’s graphical notation is more stable since it is attribute-free; in other words, object types and value types are both treated as concepts. This makes ORM immune to changes that cause attributes to be remodeled as object types or relationships.

Compared with other modeling notations, ORM diagrams can be automatically verbalized into pseudo natural language sentences. For example, the mandatory constraint in figure 4.4 is verbalized as: “*Each Professor must WorksFor at least one University*”. The subset constraint in figure 4.18 is verbalized as: “*If a Person Drives a Car then this Person must be AuthorizedWith a DrivingLicense*”. Additional explanation can be found in [JKD06] and [HC06], which provide sophisticated and multilingual verbalization templates. From a methodological viewpoint, this verbalization capability simplifies communication with non-IT domain experts and allows them to better understand, validate, and build ORM diagrams. It is worthwhile to note that ORM is the historical successor of NIAM (Natural Language Information Analysis Method), which was explicitly designed (in the early 70’s) to play the role of a stepwise methodology, that is, to arrive at the ”semantics” of a business application’s data based on natural language communication.

Indeed, the graphical expressiveness and the methodological and verbalization capabilities of ORM make it a good candidate for a graphical notation for modeling and representing ontologies and their underpinning logic.

ORM’s formal specification and semantics are well-defined ([Hal89, vBtHvdW91, tH93, dT96, dTM95]). The most comprehensive formalization in first-order logic (FOL) was carried out by Halpin in [Hal89]. Later on, some specific portions of this formalization were reexamined, such as subtypes [HP95], uniqueness [Hal02], objectification[Hal05], and ring constraints [Hal01]. Since reasoning on first order logic is far complex, namely undecidable[BCM$^+$03], the above formalizations do not enable automated reasoning on ORM diagrams, which comprises e.g. detection of constraint contradictions, implications, and inference.

To enable such automated reasoning, we map all ORM primitives and constraints into the $\mathcal{DLR}$ and the $\mathcal{SHOIN}$ description logics, which are decidable fragments of first-order logic. Our mapping is based on the ORM syntax and semantics specified in [Hal89] and [Hal01].

The remainder of the chapter is organized as follows. In section 4.0.2, we give a quick overview of description logics. Section 4.1 presents the ORM semantics in detail as well as its mapping into $\mathcal{DLR}$, and section 4.2 maps ORM into $\mathcal{SHOIN}$/OWL. In section 4.3, we illustrate the implementation of this formalization as an extension to DogmaModeler, and in section 4.4 we relate our mappings to related work. Finally, the conclusions and directions for future work are presented in section 4.5.

Remark: In this chapter, we focus only on the *logical* aspects of reusing ORM for ontology modeling. The conceptual aspects (i.e. ontology modeling verses data modeling) are discussed in [JM07, Jar05, JDM1, JM02, Jar06b], while a real-life case study that uses the ORM notation (for developing an eBusiness ontology) can be found in [JVM03].

4.0.2 Description Logics

Description logics are a family of knowledge representation formalisms. Description logics are decidable fragments of first-order logic, associated with a set of automatic reasoning procedures. The basic primitives of a description logic are the notion of a concept and the notion of a relationship. Complex concept and relationship expressions can be built from atomic concepts and relationships. For example, one can define $HumanMother$ as $Female \sqcap \exists HasChild.Person$. The expressiveness of a description logic is characterized by the set of constructors it offers. The simplest description logic is called $\mathcal{FL}^-$, which offers only the intersection of concepts, value restrictions, and a simple form of existential quantification. In other words, a *TBox* in $\mathcal{FL}^-$ is built as a set of inclusion assertions of the following forms: $C, D \ \rightarrow A \mid C \sqcap D \mid \forall R.C \mid \exists R$.

A more practical and expressive description logic is called $\mathcal{AL}$ [BCM$^+$03], and it serves as the basis for several description logics. $\mathcal{AL}$ offers a concept to be formed as ($C, D \ \rightarrow \ A \mid \top \mid \bot \mid \neg A \mid C \sqcap D \mid \forall R.C \mid \exists R.\top$), where C and D denote concept descriptions, A and R stand for atomic concept and atomic role (i.e. binary relation) respectively, $\top$ and $\bot$ denote a universal concept and a bottom concept respectively, $\neg A$ stands for atomic negation, $C \sqcap D$ stands for an intersection, $\forall R.C$ stands for a value restriction, and $\exists R.\top$ indicates limited existential quantification. [BCM$^+$03] provides more information on description logic and its applications, while [DLc] provides an online tutorial course on description logics.

$\mathcal{SHOIN}$ is an expressive description logic [HST99]. It is the logic underpinning OWL, the standard (W3C recommendation) Ontology Web Language. $\mathcal{SHOIN}$ was developed as a compromise between expressive power and decidability. The $\mathcal{SHOIN}$ syntax is described as follows. If C and D are concepts and R is a binary relation (also called role), then $(C \sqcap D)$, $(C \sqcup D)$, $(\neg C)$, $(\forall R.C)$, and $(\exists R.C)$ are also concepts. If R is simple (i.e., neither transitive nor has any transitive sub relations), then $(\leq nR)$ and $(\geq nR)$ are also concepts, where n is a non-negative integer. For C and D (possibly complex) concepts, $C \sqsubseteq D$ is called a general concept inclusion. $\mathcal{SHOIN}$ also allows hierarchies of relations ($R \sqsubseteq S$), transitivity (R_+), and inverse ($S \sqsubseteq R^-$). A recent extension [HKS06] of $\mathcal{SHOIN}$ (called $\mathcal{SROIQ}$) enables representing more advanced constructs on relations, such as complex inclusions (e.g. $S \circ R \sqsubseteq R$), disjointness, negation ($\neg R$), and local reflexivity ($\exists R.Self$). Please refer to [HST99] for the semantics of $\mathcal{SHOIN}$, and [HKS06] for more details on $\mathcal{SROIQ}$.

$\mathcal{DLR}$ (or specially its extension $\mathcal{DLR}_{ifd}$[CGL01]) is an expressive description logic. It was developed to allow the majority of the primitives and constraints used in database modeling to be represented [CDGL98a] [CLN99], including n-ary relations, identification, and functional dependencies. The basic constructs of $\mathcal{DLR}$ are concepts and n-ary relations ($n \geq 2$). Let A denote an atomic concept, and P an atomic n-ary relation. Arbitrary concepts, denoted by C and arbitrary relations denoted by R, can be built according to the following syntax respectively:

$$\begin{array}{rcl} C & ::= & \top_1 \mid A \mid \neg C \mid C_1 \sqcap C_2 \mid (\leq k[i]R) \\ R & ::= & \top_n \mid P \mid (i/n : C) \mid \neg R \mid R_1 \sqcap R_2 \end{array}$$

where P, R, R_1 and R_2 are n-ary relations ($n \geq 2$), i/n denotes the ith component of an n-ary relationship (i is an integer between 1 and n), and k denotes a non-negative integer. Relations in $\mathcal{DLR}$ are *well-typed*, which means that only relations of the same arity n can be used in expressions like $R_1 \sqcap R_2$ and $i \leq n$ whenever i denotes a component of a relation of arity n. The following are abbreviations: $\bot$ for $\neg\top_1$; $C_1 \sqcup C_2$ for $\neg(\neg C_1 \sqcap \neg C_2)$; $C_1 \Rightarrow C_2$ for $\neg C_1 \sqcup C_2$; $(\leq k[i]R)$ for $\neg(\leq k-1\ [i]R)$; $\exists[i]R$ for $(\geq 1[i]R)$; $\forall[i]R$ for $\neg\exists[i]\neg R$; and $(i : C)$ for $(i/n : C)$ if n is clear from the context.

The semantics of $\mathcal{DLR}$ is specified as follows. An *interpretation* I is constituted by an interpretation domain $\triangle^I$, and an interpretation function $.^I$ that assigns to each concept C a subset C^I of $\triangle^I$ and to each R of arity n a subset R^I of $(\triangle^I)^n$. $t[i]$ denotes the i-th component of tuple t.

$$\begin{array}{rclrcl} \top_n^I & \subseteq & (\triangle^I)^n & \top_1^I & = & \triangle^I \\ P^I & \subseteq & \top_n^I & A^I & \subseteq & \triangle^I \\ (i/n : C)^I & = & \{t \in \top_n^I | t[i] \in C^I\} & (\neg C)^I & = & \triangle^I \backslash C^I \\ (\neg R)^I & = & \top_n^I \backslash R^I & (C_1 \sqcap C_2)^I & = & C_1^I \cap C_2^I \\ (R_1 \sqcap R_2)^I & = & R_1^I \cap R_2^I & (\leq k[i]R)^I & = & \{a \in \triangle^I |\ \sharp\{t \in R^I | t[i] = a\} \leq k\} \end{array}$$

A $\mathcal{DLR}$ *TBox* comprises a finite set of inclusion assertions, where each assertion has the form: $C_1 \sqsubseteq C_2$ or $R_1 \sqsubseteq R_2$, with R_1 and R_2 of the same arity. Beside these inclusion assertions in $\mathcal{DLR}$, $\mathcal{DLR}_{ifd}$ allows the following assertions for identification **id** and functional dependencies **fd** to be expressed.

$(\mathbf{id}\ \ C\ \ [i_1]R_1, ..., [i_h]R_h)$
$(\mathbf{fd}\ \ R\, i_1, ..., i_h \rightarrow j)$
which have the following semantics.

An interpretation I satisfies the assertion $(\mathbf{id}\ \ C\ \ [i_1]R_1, ..., [i_h]R_h)$ if for all $a, b \in C^I$ and for all $t_1, s_1 \in R_1^I, ..., t_h, s_h \in R_h^I$ we have that:

An interpretation I satisfies the assertion $(\mathbf{fd}\ \ R\, i_1, ..., i_h \rightarrow j)$ if for all t, $s \in R^I$, we have that: $t[i_1] = s[i_1], ..., t[i_h] = s[i_h]$ implies $t[j] = s[j]$

Furthermore, another useful extension that has been recently included in $\mathcal{DLR}$*-Lite* [CDGL+06] which we shall use in this paper, is inclusion between projections of relations, which has the following form:

$R_2[r_{j_1}, ..., r_{j_k}] \ \sqsubseteq\ R_1[r_{i_1}, ..., r_{i_k}]$

This inclusion, the identification **id**, and the functional dependencies **fd** shall be explained in more details later in this paper. A $\mathcal{DLR}$ ABox is constituted by a finite set of assertions, called ABox assertions, of the following types:

$C(x) \quad R(x_1, ..., x_n) \quad x \neq y \quad x = y$

4.1 The formalization of ORM using $\mathcal{DLR}_{ifd}$

In this section we present the formalization of all ORM constructs using $\mathcal{DLR}_{ifd}$. The semantics of each construct is presented in detail and mapped into $\mathcal{DLR}_{ifd}$ and clear motivations for the mapping choices are given.

4.1.1 Object-Types

ORM allows a domain to be modelled by using object types that play certain roles. There are two kinds of object-type in ORM: Non-Lexical Object-Types (NOLOT) and Lexical Object-Types (LOT). Both object-types are shown as ellipses in ORM's notation, with a LOT being depicted as a dotted-line ellipse and a NOLOT as a solid-line ellipse (see figure 4.1). The difference between a LOT and a NOLOT is linguistic [VvB82]. While lexical object-types correspond to utterable entities, such as names or titles, non-lexical object-types refer to non-utterable entities, such as a person, a university, or a book. In other words, when someone says "I am John", he actually means "I am the person who is referred to by the name John". In data models, non-lexical object types are represented (and referenced) by lexical object types[1].

We represent both NOLOTs and LOTs as classes in $\mathcal{DLR}$. To distinguish between

[1]Although they are not exactly similar, the notions of LOT and NOLOT in ORM can be, for the sake of simplicity, compared to the concepts of 'Attribute' and 'Class' in UML, or the notions of 'ValueProperty' and 'ObjectProperty' in OWL

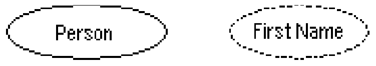


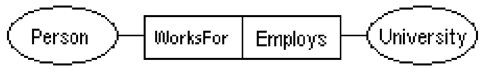
Figure 4.1: Object Types (NOLOT and LOT.) in ORM

NOLOT and LOT in a $\mathcal{DLR}$ knowledge base, we introduce four classes: LEXICAL, STRING, NUMBER, and BOOLEAN. The class LEXICAL is considered to be a supertype of the other three classes, while the other three classes are considered to be disjoint. Unless specified, each LOT is mapped by default into the class STRING. We shall return to this issue later.

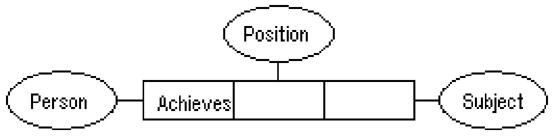
4.1.2 Roles and relationships

ORM supports n-ary relationships, where $n \geq 1$. Each argument of a relationship in ORM is called a *role*. In figure 4.2, we show examples of binary and ternary relationships. For example, the binary relationship has two roles, $WorksFor$ and $Employs$. The formalization of the general case of an ORM n-ary relationship is [Hal89]: $\forall x_1...x_n(R(x_1...x_n) \rightarrow A_1(x_1) \wedge ... \wedge A_n(x_n))$.

$\mathcal{DLR}$ supports n-ary relationships, where $n \geq 2$. Each argument of a relationship in $\mathcal{DLR}$ is called a *component* [BCM+03]. As shown in figure 4.2, we represent a relationship in ORM as a relationship in $\mathcal{DLR}$; thus, a role in ORM is seen as a component of a relationship in $\mathcal{DLR}$. Notice that the notion of 'role' in $\mathcal{DLR}$ is synonym to a relationship. Hence to avoid any confusion, the term 'role' is used in this chapter to indicate only a component of a relationship, i.e. a ORM role. Notice also that while ORM supports unary relations, $\mathcal{DLR}$ does not. For this case, we introduce a special treatment in the next section. For people who are familiar with ORM, the formalization of ORM



$$R \sqsubseteq (WorksFor : Person) \sqcap (Employs : University)$$

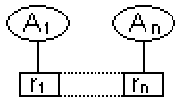


$$R \sqsubseteq (Achieves : Person) \sqcap (r2 : Position) \sqcap (r3 : Subject)$$

Figure 4.2: Examples of Binary and Ternary relations in ORM.

roles and relationships shown in figure 4.2 seems to be trivial. However, people who are familiar with description logics may not find it intuitive. This is because, unlike ORM, the components of relationships in description logics are typically not used and do not have linguistic labels. For example, one expects to see the binary relationship in Figure 4.2 represented in description logic as, $Person \sqsubseteq \forall WorksFor.University$, and $University \sqsubseteq \forall Employs.Person$. In this case, both $WorksFor$ and $Employs$ are two different relationships. This formalization requires an additional axiom to state that both relations are inverse to each other: $WorksFor \sqsubseteq Employs^{-}$.

ORM schemes formalized in this way are not only lengthy, but also become more complex when relationships other than binary are introduced. As will be shown later, our method of formalizing ORM roles and relationships will make the formalization of ORM constraints more intuitive and elegant. Rule-1 presents the general case formalization of ORM n-ary relations, where $n \geq 2$.



n ≥2

$R \sqsubseteq (r_1 : A_1) \sqcap ... \sqcap (r_n : A_n)$ ***(Rule-1)***

Remark: When mapping an ORM schema into a $\mathcal{DLR}$ knowledge base:

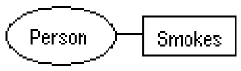
- Each role in the ORM schema should have a unique label within its relationship.
- In case a role label is null, an automatic label is assigned, such as r_1, r_2, etc.
- In case of a relationship having the same labels of its roles, such as $ColleagueOf/$ $ColleagueOf$, new labels are assigned to these roles, such as: $ColleagueOf - r_1$, $ColleagueOf - r_2$.
- Usually, ORM relationships do not have labels; thus, a unique label is automatically assigned, such as: R_1, R_2, etc.

One may notice that according to the $\mathcal{DLR}$ syntax (see section 4.0.2), a role i in an n-ary relation R is identified by a number: $[i]R$, where i is an integer. However, we use the linguistic labels of roles instead of such numbers. As stated in the previous remark, each role label in ORM is scoped within its relationship, and the concatenation of a role label and a relationship name, such as $[WorksFor]R1$, is unique in the whole ORM schema.

4.1.3 ORM unary roles

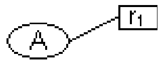
Unlike $\mathcal{DLR}$, ORM allows the representation of unary relations (see figure 4.3). The relationship in Figure 3 means that a person may smoke; in other words, any individual that 'smokes' must be a person. In first-order logic, this fact can be formalized [Hal89] as: $\forall x(Smokes(x) \rightarrow Person(x))$. One may notice that this knowledge is typically represented in description logic using the subsumption relationship, e.g. Smoker Is-A Person. However, ORM allows this knowledge to be represented using unary roles, so as to enable such roles to participate in constraints that might be defined on multiple roles or relationships[2], as shall be shown later.

To formalize ORM unary roles in $\mathcal{DLR}$, we avoid using the subsumption relationship for the reasons mentioned above. Instead, we introduce a new class called BOOLEAN, which can take on two values: either TRUE or FALSE. Each ORM unary fact is seen as a binary relationship in $\mathcal{DLR}$, where the second concept is BOOLEAN. Recall from the previous section that the class BOOLEAN is a subclass of LEXICAL, and that it is disjoint with the classes STRING and NUMBER. Rule-2 presents the general case formalization of ORM unary fact types. As shall be shown in the following sections, formalizing unary facts in this way allows the simple and elegant formalization of complex constraints that might be defined on multiple relations.



$$R \sqsubseteq (Smokes : Person) \sqcap (r2 : BOOLEAN)$$

Figure 4.3: Example of Unary relations.



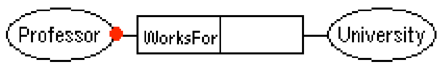
$R \sqsubseteq (r_1 : A) \sqcap (r2 : BOOLEAN)$ ***(Rule-2)***

4.1.4 Role Mandatory

There are two types of mandatory constraints in ORM: role mandatory, and disjunctive mandatory.

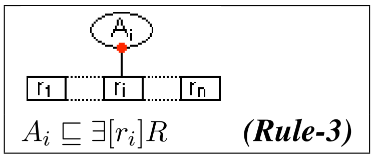
The role mandatory constraint in ORM is depicted as a dot on the line connecting a role with an object type (see figure 4.4). It is used to constrain a role played by an object type, such that each instance of that object type must play this role at least once. The example in figure 4.4 indicates that, in every interpretation of this schema, each instance of the object-type Professor must work for at least one University. Rule-3 presents the general case formalization of the role mandatory constraint.

[2]In our opinion, using unary roles instead of a subsumption relationship has other ontological advantages. Ontologically, it is not completely accurate to say that a "smoker" is a type of person. A smoker is a *role* that a person might play, rather than a *type*. See [GW02] for more details on 'type' versus 'role' in ontology engineering.



$Professor \sqsubseteq \exists[WorksFor]R$

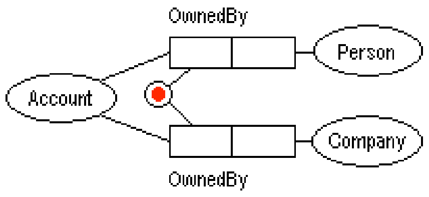
Figure 4.4: Example of a role mandatory constraint.



$A_i \sqsubseteq \exists[r_i]R$ **(Rule-3)**

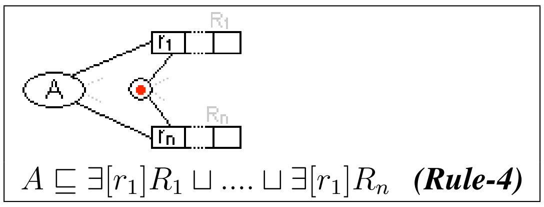
4.1.5 Disjunctive Mandatory

The disjunctive mandatory constraint is used to constrain a set of two or more roles connected to the same object type. It means that each instance of the object type's population must play at least one of the constrained roles. For example, the disjunctive mandatory in figure 4.5 means that each account must be owned by at least a person, a company, or both. Rule-4 presents the general case formalization of a disjunctive mandatory constraint.



$R1 \sqsubseteq (OwnedBy : Account) \sqcap (r2 : Person)$
$R2 \sqsubseteq (OwnedBy : Account) \sqcap (r2 : Company)$
$Account \sqsubseteq \exists[OwnedBy]R1 \sqcup \exists[OwnedBy]R2$

Figure 4.5: Example of a disjunctive mandatory constraint.

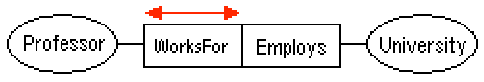


$A \sqsubseteq \exists[r_1]R_1 \sqcup .... \sqcup \exists[r_1]R_n$ **(Rule-4)**

4.1.6 Role Uniqueness

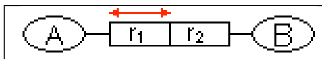
We distinguish between three types of uniqueness constraints in ORM: role uniqueness, predicate uniqueness, and external uniqueness.

Role uniqueness is represented by an arrow spanning a single role in a binary relationship. As shown in figure 4.6, this uniqueness constraint states that, in every interpretation of this schema, each instance of Professor must work for at most one University, i.e. each occurrence is unique. Rule-5 presents the general case formalization of the role uniqueness constraint.



$Professor \sqsubseteq (\leq 1[WorksFor]R)$

Figure 4.6: Example of a role uniqueness constraint.



$A \sqsubseteq (\leq 1[r_i]R)$ ***(Rule-5)***

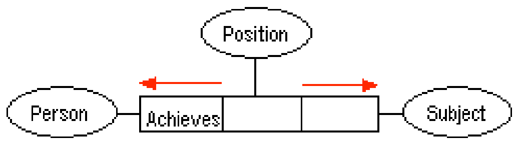
4.1.7 Predicate Uniqueness

An arrow spanning more than a role in a relationship of arity n represents *predicate uniqueness*. As shown in figure 7, this uniqueness constraint states that, in any population of this relationship, the person and subject pair must be unique together. The general case of this constraint is formalized in [Hal89] as: $\forall x_1, .., x_i, .., x_n, y(R(x_1, .., x_i, .., x_n) \wedge R(x_1, .., y, x_{i+1}, .., x_n) \rightarrow x_i = y)$.

We formalize this uniqueness constraint using the notion of *functional dependency* **fd** in $\mathcal{DLR}_{ifd}$ [CGL01], which has the form:

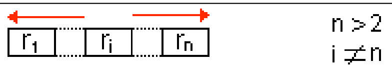
$$(\textbf{fd} \quad R\, r_1, ..., r_h \rightarrow r_j)$$

where R is a relation, and $r_1, ..., r_h$, r_j denote roles in R. The notion of functional dependency requires two tuples of a relationship that agree on the constrained components $r_1, ..., r_h$ to also agree on the un-constrained component r_j. The set of the constrained roles (on the the left-side of the **fd** assertion) uniquely determines the un-constrained role (which is on the the right side of the assertion). In other words, we say that the un-constrained role is functionally dependent upon the set of the constrained roles. We call the set of the constrained roles *determinant* and the un-constrained role *dependent*. The following is the general case formalization rule of a predicate uniqueness constraint.



$R \sqsubseteq (Achieves : Person) \sqcap (r2 : Position) \sqcap (r3 : Subject)$
$\textbf{fd}\ R\ Achieves, r_2 \rightarrow r_3$

Figure 4.7: Example of an ORM predicate uniqueness constraint.



$\textbf{fd}\ R\, r_1, ..., r_{i-1}, r_{i+1}, ..., r_n \rightarrow r_i$ ***(Rule-6)***
(Not supported by any DL reasoner yet)

Notice that our formalization excludes the following cases:

- Role uniqueness in a binary relationship: Although it is theoretically possible to use the above formalization in case of a binary relationship, we keep the formalization of this case separate (see rule-5) for implementation reasons. This is because: 1) rule-5 is supported in most description logic reasoners while rule-6 is not implemented in any reasoner yet, and 2) reasoning on functional dependencies cannot be performed on TBox only. In other words, as functional dependencies in $\mathcal{DLR}_{ifd}$ are seen as extra assertions (i.e. outside the TBox), the reasoning process to check whether the **fd** assertions are violated is reduced to ABox satisfiability. If there is no ABox, one cannot reason over the **fd** assertions.

- A single role uniqueness in an $n-$ary relationship where ($n > 2$), since it is always a non-elementary fact type (see [Win90], chapter 12): This case is considered an illegal constraint in ORM (see [Hal01], chapter 4), with [CGL01] proving that it leads to undecidability in reasoning. Therefore, this case is ruled out in our formalization.

- A uniqueness constraint spanning all roles of a relationship, where $i = n$ (see figure 8): This uniqueness states that the population of the entities Person and University together is unique in each tuple of the relationship. Notice that there is no logical significance to such a constraint, i.e. it is not a constraint [Hal89]. This constraint is used in databases to prevent data redundancy, which is not important in logic. For a relationship of arity n to be elementary, any uniqueness constraint on it must span at most $n - 1$ roles [Hal89]. However, this constraint can be easily formalized by objectifying the relation R and then using the notion of **id**, as: (**id** $R\ [WorksFor]R, [Employs]R$).

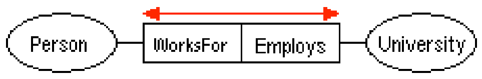
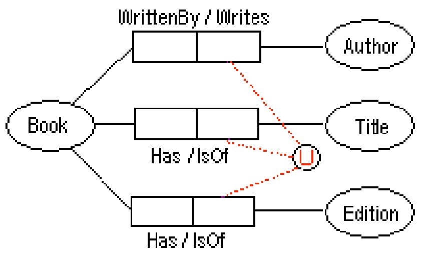


Figure 4.8: Example of the uniqueness constraint spanning all roles.

4.1.8 External Uniqueness

External uniqueness constraints (denoted by "U") apply to roles from different relationships. The roles that participate in such a uniqueness constraint uniquely refer to an object type. For example, as shown in figure 9, the combination of (Author, Title, Edition) must be unique. In other words, different values of (Author, Title, Edition) refer to different Books. Formalizing this constraint in description logic is possible using the



$$R1 \sqsubseteq (WrittenBy : Book) \sqcap (Writes : Author)$$
$$R2 \sqsubseteq (Has : Book) \sqcap (IsOf : Title)$$
$$R3 \sqsubseteq (Has : Book) \sqcap (IsOf : Edition)$$
$$\textbf{id}\ \ Book\ \ [WrittenBy]R1 \sqcup [Has]R2 \sqcup [Has]R3$$

Figure 4.9: Example of the role of an external uniqueness constraint on different relationships.

notion of identity **id** in $\mathcal{DLR}_{ifd}$ [CGL01]. In case the external uniqueness is defined on binary relationships and the common concept to be constrained is directly connected to these relations, the formalization is direct (see figure 9). In other cases, the formalization becomes more complex. We shall try to simplify and explain this complexity in the following section.

The notion of identity **id** in $\mathcal{DLR}_{ifd}$ has the form:

$$(\textbf{id}\ \ C\ \ [r_1]R_1, ..., [r_n]R_n)$$

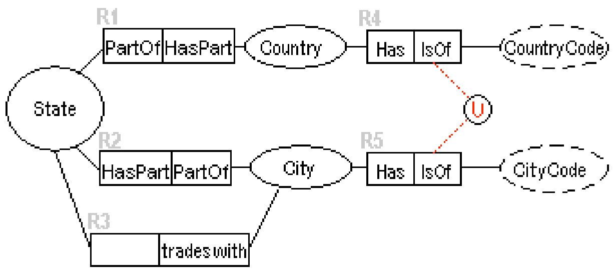
Where C is a concept, each R_i is a relation, and each r_i is a role in R_i that is connected to C. The identity **id** in $\mathcal{DLR}_{ifd}$ states that two instances of the concept C cannot agree on the participation in $R_1, ..., R_n$ via their roles $r_1, ..., r_n$, respectively. See [CGL01] for more details on this.

In ORM, the intuition of external uniqueness is that the combination of $r_1, ..., r_n$ in $R_1, ..., R_n$ respectively must be unique. The formalization of the general case [Hal89] of this constraint (see the figure in rule-7) is: $\forall x_1, x_2, y_1..y_n(R_1(x_1, y_1) \wedge ... \wedge R_n(x_1, y_n) \wedge (R_1(x_2, y_1) \wedge ... \wedge R_n(x_2, y_n) \rightarrow x_1 = x_2)$.

This allows one to define uniqueness on roles that are not directly "connected" to a *common concept*. For example, although the external uniqueness in figure 10 means that the combination of {CountryCode, CityCode} must be unique, it does not tell us that the combination is unique for which concept. In other words, the notion of "common concept" is not explicitly regarded, neither in the ORM graphical notation nor in its underlying semantics [Hal89] [Hal02] [vBtHvdW91]. This is the reason why the notation of external uniqueness is drawn on the roles that are connected with a unique combination, and not on the roles of a common concept. For example, the external uniqueness constraint in figure 10 is drawn on the {[IsOf]R4, [IsOf]R5} roles, even though for the notion of **id** the involved roles should be {[Has]R4, [Has]R5}. To interpret the external uniqueness (i.e. the semantics) in figure 10, a join path should be performed on $R4 - R1$ and $R5 - R2$. In other words, although the notion of "common concept" does not exist in ORM, it is assumed that there must be a join path between the constrained roles. If this path cannot be constructed, then the external uniqueness is considered illegal [Hal02], i.e. an error in the ORM schema.

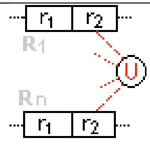
This freedom in ORM (i.e. being able to draw an external uniqueness constraint on arbitrary roles) leads to the ambiguity of the paths on which a join can be performed [Hal02]. For example, should it be on $\{R4 - R1, R5 - R2\}$ or $\{R4 - R1, R5 - R3\}$? The construction of such join paths becomes more complex (even for human eyes) in large schemes or when objectified (i.e. reified) predicates are involved. Figure 11 show some cases of complex external uniqueness.

We formalize the general case of external uniqueness using the notion of **id** in $\mathcal{DLR}_{ifd}$, but we use the concept Top as the common concept C (see rule-7). As shown in figure 9, the formalization (using Top) means that *any* two individuals must agree on their participation in roles: [WrittenBy]R1, [Has]R2 and [Has]R3.



id $Top\ [Has]R4 \sqcup [Has]R5$

Figure 4.10: Example of an external uniqueness constraint through multiple join paths, adopted from [Hal02].



id $Top\ [r_1]R_1, ..., ...[r_1]R_n$ ***(Rule-7)***
(Not supported by any DL reasoner yet)

Although the use of the Top concept yields a simple and elegant formalization, intensive ABox reasoning may be required. In practice, we recommend using the Uniquest algorithm [vdWtHvB92]. This algorithm is designed to compute the shortest join path connecting the constrained roles for an external uniqueness constraint, no matter what its level of complexity is. The result is a *derived relation*, which represents the shortest join path. This derived relation can then be used instead of the concept Top in rule-7.

4.1.9 Role Frequency Constraints

We distinguish between frequency constraints that span 1) a single role, which we call "role frequency" constraints, and 2) multiple roles, which we call "multiple-role fre-

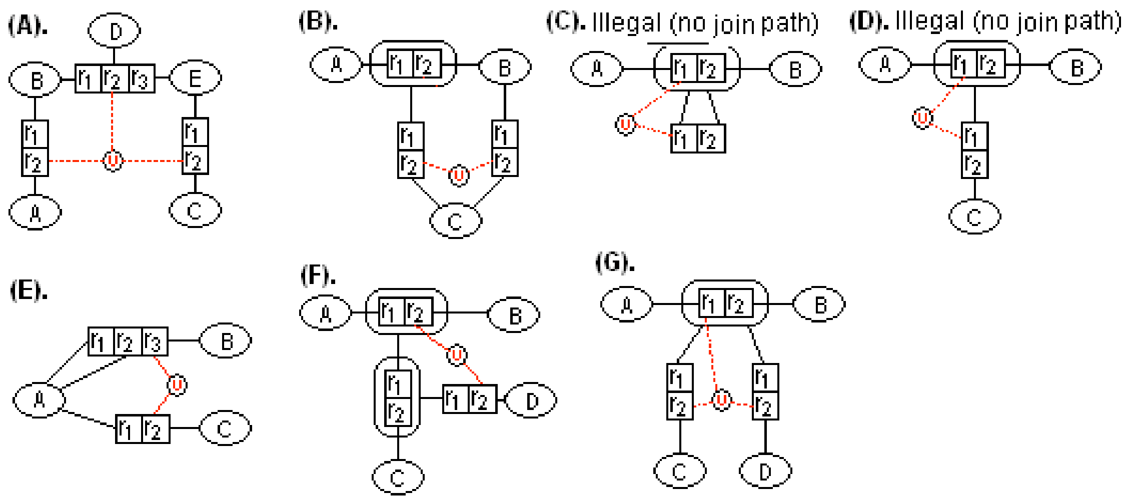
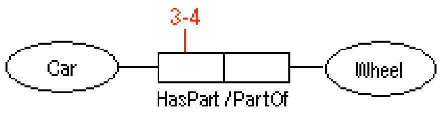


Figure 4.11: Examples of complex external uniqueness in ORM, adopted from [vdWtHvB92].

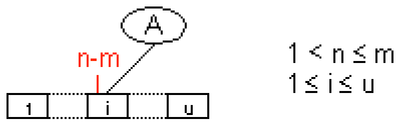
quency" constraints.

A role frequency constraint $(min - max)$ is used to specify the number of occurrences that this role can be played by its object-type. A frequency constraint on the ith role of an n-ary relation is formalized [Hal89] as: $\forall x[x \in R.i \rightarrow \exists^{n,m} z(R(z) \wedge z_i = x)]$. For example, the frequency constraint in figure 12 indicates that if a car has wheels, it must have at least 3 and at most 4 wheels. Notice that a frequency constraint of 1, which is equivalent to an internal uniqueness constraint on this role, is covered by rule-5. The example here says that if it happens that a car has wheels (although it may not have them), then it must have at least 3 and at most 4 wheels. Therefore, we formalize this constraint by conjugating $\bot$ to the $(min - max)$ cardinality, i.e. either there is no occurrence, or it must be within the $(min - max)$ range, which is the exact meaning in ORM. Rule-8 presents the general case mapping rule of a role frequency constraint.



$Car \sqsubseteq \exists^{\geq 3, \leq 4}[HasPart]R \sqcup \bot$

Figure 4.12: Example of a role frequency.



$A \sqsubseteq \exists^{\geq n, \leq m}[i]R \sqcup \bot$ ***(Rule-8)***

4.1.10 Multiple-role Frequency Constraints

A multiple-role frequency constraint spans more than one role (see figure 13). This constraint means that, in the population of this relationship, A and C must occur together (i.e. as a tuple) at least 3 times and at most 6 times. Notice that a frequency constraint of 1 on a set of roles is covered by rule-6, which is equivalent to a predicate uniqueness on these role. Up to our knowledge, such a cardinality constraint cannot be formalized in description logic. However, this constraint is extremely rare in practice, [Hal01] presents an example of this constraint and shows that it can be remodeled and achieved by a combination of uniqueness and single-role frequency constraints, which are indeed cheaper to compute and reason about. *Exception-1* presents the general case of the ORM multiple-role frequency constraint and its formalization in first order logic [Hal89].

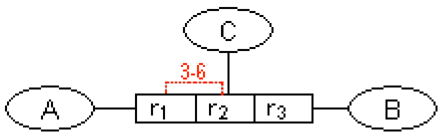
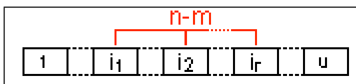


Figure 4.13: A Multiple-Role Frequency Constraint.

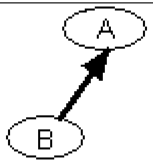


Not supported in DL ***(Exception-1)***

$\forall x\ [R(x) \rightarrow \exists^{n,m} z(R(z) \wedge z_{i1} = x_{i1} ... \wedge z_{ir} = x_{ir})]$

4.1.11 Subtypes

Subtypes in ORM are proper subtypes. For example, we say that B is a proper subtype of A if and only if the population of B is always a subset of the population of A, and A $\neq$ B. This implies that the subtype relationship is acyclic; hence, loops are illegal in ORM. To formalize this relationship in $\mathcal{DLR}$, we introduce an additional negation axiom for each subtype relation. For example, Subtype(Man, Person) in ORM is formalized as: $(Man \sqsubseteq Person) \sqcap (Person \not\sqsubseteq Man)$. Rule-9 presents the general case formalization of ORM subtypes. Notice that "$\not\sqsubseteq$" is not part of the $\mathcal{DLR}$ syntax. However, it can be implemented by reasoning on the ABox to make sure that the population of A and the population B are not equal.



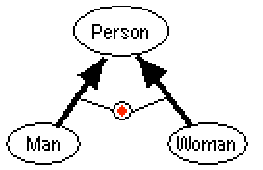
$B \sqsubseteq A$
$A \not\sqsubseteq B$ ***(Rule-9)***

Remarks. First: In ORM, object-types are mutually exclusive by default, unless they share a common type. A subtype hierarchy corresponds to a directed acyclic graph with a *unique top*. A specialization hierarchy can thus considered to be a semi-lattice, where the least upper bound should exist for each pair of subtypes in the same hierarchy. Refer to [HP95] and ([Hal01], chapter 6) for more details. This "unique top" property of ORM subtypes can be implemented by stating that all object types in the schema, except those that have a subtype link among themselves, are disjoint.

Second: Subtypes in ORM should be *well-defined*, which means that users should introduce some rules explicitly to define a subtype. Such definitions are not part of the graphical notation and are typically written in the FORMAL langauge [Hal89]. The idea of the ORM FORMAL language is similar to the idea the OCL language for UML. For example: if one states that (Man Is-a Person), then a textual rule on Man is defined e.g. *"who has Gender='Male"'*. Since such rules are not part of the graphical notation, we do not include them in our formalization. We assume that textual rules that are not part of the ORM graphical notation are written in $\mathcal{DLR}$ directly.

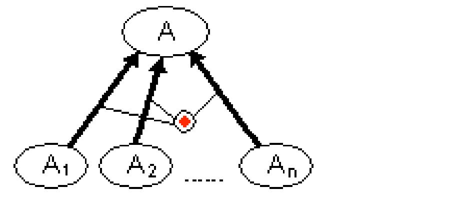
4.1.12 Total Constraint

The total constraint ($\odot$) between subtypes means that the population of the supertype is exactly the union of the population of these subtypes. For example, the constraint in figure 4.14 means: $Person = Man \cup Woman$; that is, each person must be at least a man or a woman, or both. Rule-10 presents the general case formalization of the Total constraint.



$Person \sqsubseteq Man \sqcup Woman$

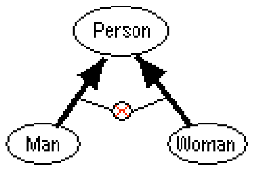
Figure 4.14: Example of Total constraint.



$A \sqsubseteq A_1 \sqcup A_2 \sqcup ... \sqcup A_n$ ***(Rule-10)***

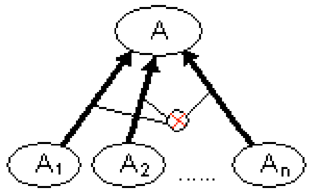
4.1.13 Exclusive Constraint

The exclusive constraint ($\otimes$) between subtypes means the population of these subtypes is distinct, i.e. the intersection of their populations must be empty. For example, the constraint in figure 4.15 means [Hal01]: $Man \cap Woman = \{\}$; that is, if a person is a woman, then this person cannot be a man at the same time. Notice that ORM allows an exclusive constraint to be drawn between more than two object types. In this case, all objects types involved are *pairwise* disjoint. A single exclusive constraint between n object types can be replaced by $n(n-1)/2$ pairs of exclusive constraints. Rule-11 presents the general case formalization of the Exclusive constraint.



$Man \sqcap Woman \equiv \perp$

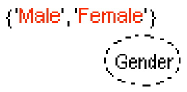
Figure 4.15: Example of an Exclusive constraint.



$(A_i \sqcap A_j \equiv \perp)$ for each $i \in \{1..n\_1\}, j \in \{i+1..n\}$ ***(Rule-11)***

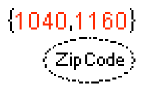
4.1.14 Value Constraints

The value constraint in ORM indicates the possible values (i.e. instances) for an object type. A value constraint on an object type A is denoted as a set of values $\{s_1, ..., s_n\}$ depicted near an object type, which indicate that $(\forall x[A(x) \equiv x \in \{s_1, ..., s_n\}])$ [Hal89] (see figure 16 and 17). Value constraints can be declared only on lexical object types, and values should be well-typed, i.e. its datatype should be either a string such as $\{'be', '39', 'it', '32'\}$ or a number such as $\{1, 2, 3\}$. Notice that quotes are used to distinguish string values from number values. As discussed earlier, if a LOT has no value constraint on it, then it is, by default, seen as a subtype of LEXICAL. If it has a value constraint, it must be a subtype of either the STRING or the NUMBER classes. Rule-12 and Rule-13 respectively present the general case formalization of the value constraints whether its datatype is a string or a number.



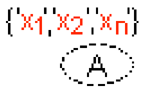
$Gender \sqsubseteq STRING$
$Gender \equiv \{Male, Female\}$

Figure 4.16: A STRING Value constraint

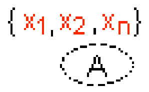


$ZipCode \sqsubseteq NUMBER$
$ZipCode \equiv \{1040, 1160\}$

Figure 4.17: A NUMBER Value constraint



$A \sqsubseteq STRING$
$A \equiv \{x_1, .., x_n\}$ ***(Rule-12)***

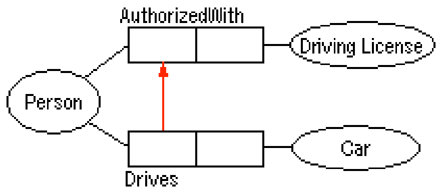


$A \sqsubseteq NUMBER$
$A \equiv \{x_1, .., x_n\}$ ***(Rule-13)***

Outlook: We plan to extend our formalization of the ORM value constraint to include other data types, such as real, integer, and boolean, which are not discussed here.

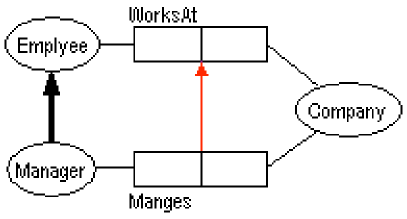
4.1.15 Subset Constraint

The subset constraint ($\rightarrow$) between roles (or sequences of roles) is used to restrict the populations of these roles, since one is a subset of the other. In figure 4.18, we illustrate a subset constraint. This indicates that each person who drives a car must be authorized by a driving license: $\forall x(x \in R2.Drives \rightarrow x \in R1.AuthorizedWith)$ [Hal89]. If an instance plays the subsuming role, then this instance must also play the subsumed role. Rule-14 presents the general case formalization of a subset constraint between two roles.



$[Drives]R2 \sqsubseteq [AuthorizedWith]R1$

Figure 4.18: A Subset between two roles.



$R2 \sqsubseteq R1$

Figure 4.19: A Subset between two relations.

A subset constraint that is declared between all roles in a relationship and all roles in another relationship implies that the set of tuples of the subsuming relation is a subset of the tuples of the subsumed relation. For example, figure 19 denotes the case that if a person manages a company, this person must also be employed in that company. Rule-15 presents the general case formalization of a subset constraint between two relations.

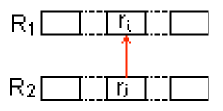
ORM also allows subset constraints between tuples of (not necessarily contiguous) roles as shown in rule-16, where each i^{th} and j^{th} roles must have the same type. The population of the set of the j^{th} roles is a subset of the population of the set of the i^{th} roles. The FOL formalization of the general case of this constraint [Hal89] is : $\forall x_1...x_k[\exists y(R_2(y) \wedge x_1 = y_{i_1} \wedge ... \wedge x_k = y_{i_k}) \rightarrow \exists z(R_1(z) \wedge x_1 = z_{j_1} \wedge ... \wedge x_k = z_{j_k})]$.

To formalize this constraint in description logic, we use the recent extension to *DLR-Lite* [CDGL+06] that allows inclusion assertions between *projections* of relations of the forms:

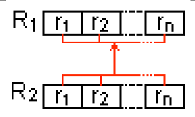
$$R_2[r_{j_1}, ..., r_{j_k}] \sqsubseteq R_1[r_{i_1}, ..., r_{i_k}]$$

where R_1 is an n-ary relation, $r_{i_1}, ..., r_{i_k} \in \{r_1, ..., r_n\}$, and $r_{i_p} \neq r_{i_q}$ if $r_p \neq r_q$; R_2 is an m-ary relation, $r_{j_1}, ..., r_{j_k} \in \{r_1, ..., r_m\}$, and $r_{j_p} \neq r_{j_q}$ if $r_p \neq r_q$.

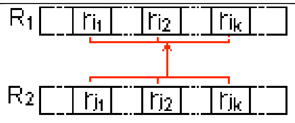
Using this extension, any ORM set-comparison constraint formalized hereafter between two sets of (not contiguous) roles becomes direct. Rule-16 shows the subset general case.



$[r_j]R_2 \sqsubseteq [r_i]R_1$ ***(Rule-14)***



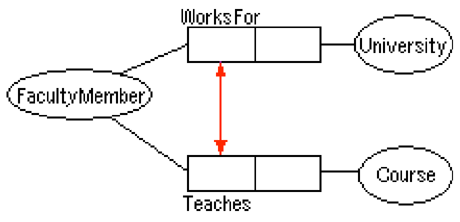
$R_2 \sqsubseteq R_1$ ***(Rule-15)***



$R_2[r_{j_1}, ..., r_{j_k}] \sqsubseteq R_1[r_{i_1}, ..., r_{i_k}]$ ***(Rule-16)***

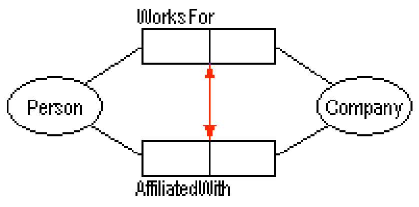
4.1.16 Equality Constraint

Similar to the subset constraint, the equality constraint ($\leftrightarrow$) can be declared between two roles, two relationships, or two sequences of roles. The equality constraint in figure 20 denotes that: Each FacultyMember WorksFor a University iff this FacultyMember Teaches a Course. The quality constraint is equivalent to two subset constraints opposed to each other. Rule-17 presents the general case formalization of an equality constraint between two roles.



$[WorksFor]R_1 \equiv [Teaches]R_2$

Figure 4.20: Equality between two roles.

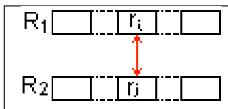


$R1 \equiv R2$

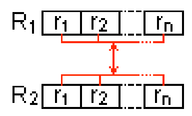
Figure 4.21: Equality between two relations.

Equality constraints can also be declared between two relationships (see figure 21). This example means that each person who works for a company must be affiliated with that company, and vise versa. Rule-18 presents the general case formalization of an equality constraint between two relationships.

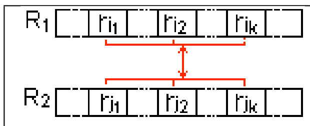
Similar to rule-16, rule-19 presents the formalization of an equality constraint between two sets of (not contiguous) roles, i.e. equivalence between two projections. The FOL formalization of this constraint [Hal89] is: $\forall x_1...x_k[\exists y(R_1(y) \wedge x_1 = y_{i_1} \wedge ... \wedge x_k = y_{i_k}) \equiv \exists z(R_2(z) \wedge x_1 = z_{j_1} \wedge ... \wedge x_k = z_{j_k})]$



$[r_i]R_1 \equiv [r_j]R_2$ ***(Rule-17)***



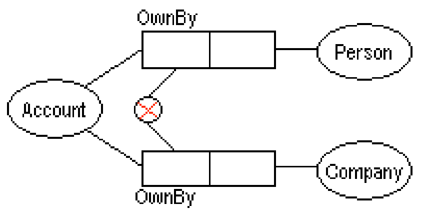
$R_1 \equiv R_2$ ***(Rule-18)***



$R_1[r_{i_1}, ..., r_{i_k}] \equiv R_2[r_{j_1}, ..., r_{j_k}]$ ***(Rule-19)***

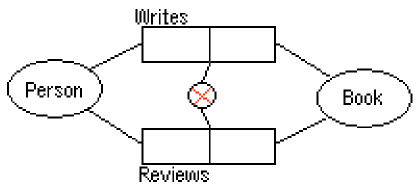
4.1.17 Exclusion Constraint

Similar to the subset and quality constraints, the exclusion constraint ($\otimes$) can be declared between roles, relationships, and sequences of (not contiguous) roles. Figure 22 shows an example of the exclusion constraint between two roles. It says that an Account cannot be Owned-By a Company and a Person at the same time. Figure 23 presents an example of the constraint between two relationships: it says that the same Person cannot Write and Review the same Book.



$[OwnedBy]R1 \sqsubseteq \neg[OwnedBy]R2$

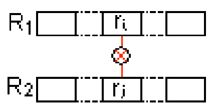
Figure 4.22: Exclusion between two roles.



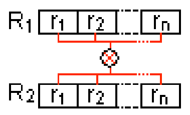
$R1 \sqsubseteq \neg R2$

Figure 4.23: Exclusion between relations.

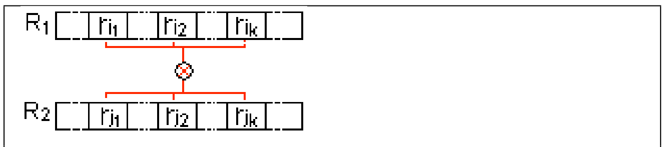
Rule-20 and rule-21 present the general cases of an exclusion constraint between two roles and relationships and its formalization in $\mathcal{DLR}$. The formalization of the exclusion constraint between two sets of (not contiguous) roles is presented in rule-22 as two distinct projections. In FOL, this constraint is formalized as [Hal89]: $\forall x_1...x_k \neg[\exists y(R_1(y) \wedge x_1 = y_{i_1} \wedge ... \wedge x_k = y_{i_k}) \wedge \exists z(R_2(z) \wedge x_1 = z_{j_1} \wedge ... \wedge x_k = z_{j_k})]$



$[r_i]R_1 \sqsubseteq \neg[r_j]R_2$ ***(Rule-20)***



$R_1 \sqsubseteq \neg R_2$ ***(Rule-21)***



$R_1[r_{i_1},...,r_{i_k}] \sqsubseteq \neg R_2[r_{j_1},...,r_{j_k}]$ ***(Rule-22)***

Remark: ORM allows the exclusion constraint to be drawn among more than two sets of roles; this is called “Multiple Exclusion” (see the left-side of figure 24). The idea of this redrawing merely is to avoid exhaustively marking exclusion constraints between each pair of roles. This case can be converted into pairs of exclusions as shown in the right-side. A single exclusion constraint across n roles replaces $n(n-1)/2$ separate exclusion constraints between two roles [Hal01].

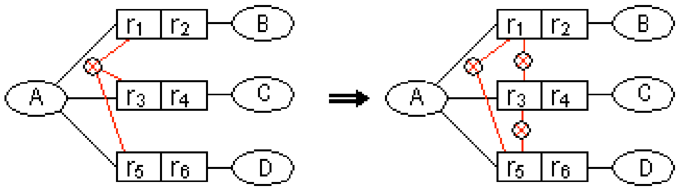
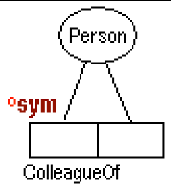


Figure 4.24: Multiple verses pair exclusions.

In the following we formalize the Ring Constraints. ORM allows ring constraints to be applied to a pair of roles (i.e. on binary relations) that are connected directly to the same object-type, or indirectly via supertypes. Six types of ring constraints are supported by ORM: symmetric (sym), asymmetric (as), antisymmetric (ans), acyclic (ac), irreflexive (ir), and intransitive (it). Below, we present the basics of these constraints, as found in [Hal01], and their formalization in description logic.

4.1.18 Symmetric Ring Constraint (sym)

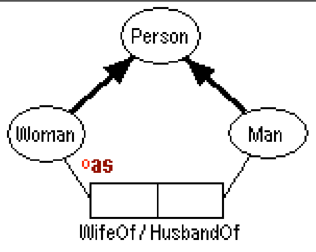
The symmetric constraint states that if a relation holds in one direction, it should also hold on the other direction, such as “colleague of” and “partner of”. R is symmetric over its population *iff* $\forall x, y[R(x,y) \longrightarrow R(y,x)]$. The example shown in rule-23 illustrates the symmetric constraint and its general case formalization in $\mathcal{DLR}$.



$R \sqsubseteq R^{-1}$ ***(Rule-23)***

4.1.19 Asymmetric Ring Constraint (as)

The asymmetric constraint is the opposite of the symmetric constraint. If a relation holds in one direction, it cannot hold on the other; an example would be “wife of” and “parent of”. R is asymmetric over its population *iff* $\forall xy, R(x, y) \longrightarrow \neg R(y, x)$ The example shown in rule-24 illustrates the asymmetric constraint and its general case formalization in $\mathcal{DLR}$.



$R \sqsubseteq \neg R^{-1}$ ***(Rule-24)***

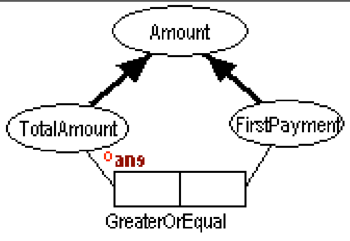
4.1.20 Antisymmetric Ring Constraint (ans)

The antisymmetric constraint is also an opposite to the symmetric constraint, but not exactly the same as asymmetric; the difference is that all asymmetric relations must be irreflexive, which is not the case for antisymmetric. R is antisymmetric over its population *iff* $\forall xy, x \neq y \wedge R(x, y) \longrightarrow \neg R(y, x)$ (see the example in rule-25).

To formalize this constraint (and some other constraints below) in description logic, we use the concept $(\exists R.Self)$ that has been introduced recently to the $\mathcal{SROIQ}$ description logic [3][HKS06]. The semantics of this concept simply is:

$$(\exists R.Self)^I = \{x \mid < x, x > \in R^I\}$$

Notice that this concept is not *yet* included in the $\mathcal{DLR}$ description logic. However, as [HKS06] show, this concept can be added without causing difficulties in reasoning. Rule-25 illustrates the antisymmetric constraint and its general case formalization.

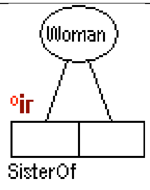


$\top \sqsubseteq (\exists R.Self) \sqcup (\neg \exists R^{-}.\top)$ ***(Rule-25)***
Not supported by any DL reasoner yet

[3] $\mathcal{SROIQ}$ is an extension of the description logics $\mathcal{SHOIN}$ [HST99] [HPSvH03] and $\mathcal{RIQ}$ [HS04].

4.1.21 Irreflexive Ring Constraint (ac)

The irreflexive constraint on a relation states that an object cannot participate in this relation with himself. For example, a person cannot be the "parent of" or "sister of" himself. R is Irreflexive over its population *iff* $\forall x, \neg SisterOf(x, x)$. As discussed above, formalizing this constraint in description logic is also possible using the concept $\exists R.Self$. Rule-26 illustrates the irreflexive constraint and its general case formalization in description logic.

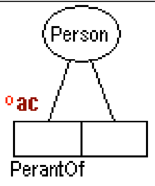


$\top \sqsubseteq \neg\exists R.Self$ ***(Rule-26)***

Not supported by any DL reasoner yet

4.1.22 Acyclic Ring Constraint (ac)

We say R is acyclic over its population *iff* $\forall x[\neg Path(x, x)]$. For example, a Person cannot be directly (or indirectly through a chain) ParentOf himself. In ORM, this constraint is preserved as a difficult constraint. "Because of their recursive nature, acyclic constraints maybe expensive or even impossible to enforce in some database systems."[Hal01]. Indeed, even some highly expressive description logics support notions such as n-tuples and recursive fixed-point structures, from which one can build simple lists, trees, etc. However, to our knowledge, acyclicity with any depth on binary relations cannot be represented.

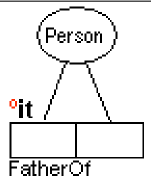


(Exception-2)

Not supported in Description logic

4.1.23 Intransitive Ring Constraint (ac)

A relation R is intransitive over its population *iff* $\forall x, y, z[R(x, y) \wedge R(y, z) \longrightarrow \neg R(x, z)]$. If Person X is FatherOf Person Y, and Y is FatherOf Z, then it cannot be that X is FatherOf Z. We formalize this constraint using the notion of *role-composition* in description logic. The composition of the two relations R and S (written as $R \circ S$) is a relation, such that: $R^I \circ S^I = \{(a, c)|\exists b.(a, b) \in R^I \wedge (b, c) \in S^I\}$. Hence, any composition with R itself ($R \circ R$) should not imply R, see rule-28.



$R \circ R \sqsubseteq \neg R$ ***(Rule-28)***

Not supported by any DL reasoner yet

The relationships between the six ring constraints described above are formalized by [Hal01] using an Eular diagram, as shown in figure 25. This formalization helps one to visualize the implication and incompatibility between these constraints. For example, one can see that acyclic implies reflexivity, intransitivity implies reflexivity, the combination of antisymmetric and reflexivity is exactly asymmetric, acyclic and symmetric are incompatible, etc.

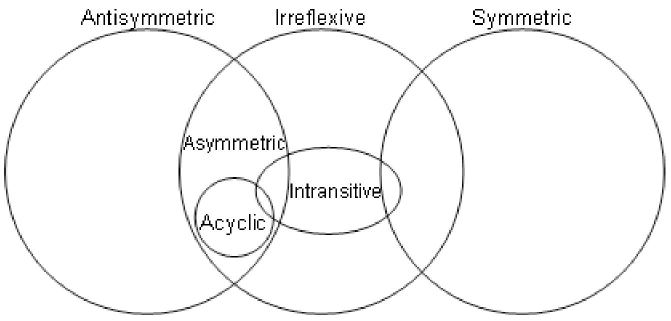
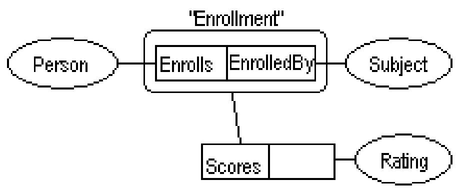


Figure 4.25: The relationships between ring constraints.

4.1.24 Objectified Relations

An objectified relation in ORM is a relation that is regarded as an object type, receives a new object type name, and is depicted as a rectangle around the relation. To help explain predicate objects in ORM, we use a familiar example (see figure 26 [Hal01]). In this example, each (Person, Subject) enrollment is treated as an object type that scores a rating. Predicate objects in ORM are also called objectified relationship types or nested fact types. The general case of predicate objects in ORM is formalized in [Hal89] as: $\forall x[A(x) \equiv \exists x_1, ..., x_n(R(x_1, ..., x_n) \wedge x = (x_1, ..., x_n))]$ In addition to this axiom, it is assumed that there must be a uniqueness constraint spanning all roles of the objectified relation, although it is not explicitly stated in the diagram. This is to indicate that e.g. each person may enroll in many subjects, and the same subject may be enrolled by many persons; see [Hal01] or the recent [Hal05] for more details.



$$
\begin{aligned}
&\mathit{Enrollement} \sqsubseteq \\
&\quad [\$1]\mathit{Enrolls} \sqcap (\leq 1[\$1]\mathit{Enrolls}) \sqcap \forall[\$1](\mathit{Enrolls} \Rightarrow (\$2 : \mathit{Student}))\sqcap \\
&\quad [\$1]\mathit{EnrolledBy} \sqcap (\leq 1[\$1]\mathit{EnrolledBy}) \sqcap \forall[\$1]((\mathit{EnrolledBy} \Rightarrow (\$2 : \mathit{Subject})) \\
&\mathbf{id}\ \ \mathit{Enrollment}\ \ [\$1]\mathit{Enrolls}, [\$1]\mathit{EnrolledBy}
\end{aligned}
$$

Figure 4.26: Example of an objectification in ORM.

Predicate objects in ORM can be formalized using the notion of *reification* in $\mathcal{DLR}_{ifd}$. Reifying an n-ary relationship into a $\mathcal{DLR}_{ifd}$ concept is done by representing this concept with n binary relations, with one relationship for each role[DBG05]. To help understand this reification, we demonstrate the "Enrollment" example (see figure 27) by remodeling the relation into two binary relations, one for the role "Enrolls" and one for the role "EnrolledBy". The new concept "Enrollment" is defined as shown in figure 26. In this definition: ($[\$1]\mathit{Enrolls}$ and $[\$1]\mathit{EnrolledBy}$) specify that the concept "Enrollment" must have all roles "Enrolls" and "EnrolledBy" of the relationship, ($\leq 1[\$1]\mathit{Enrolls}$ and $\leq 1[\$1]\mathit{EnrolledBy}$) specify that each of these roles is single-valued, and ($\forall[\$1](\mathit{Enrolls} \Rightarrow \$2 : \mathit{Student})$ and $\forall[\$1]((\mathit{EnrolledBy} \Rightarrow \$2 : \mathit{Subject})$) specify the object type each role belong to. The last identity **id** assertion is to specify a uniqueness constraint spanning all roles (i.e. "Enrolls" and "EnrolledBy"). Rule-29 presents the general case formalization of the objectified predicates in $\mathcal{DLR}_{ifd}$.

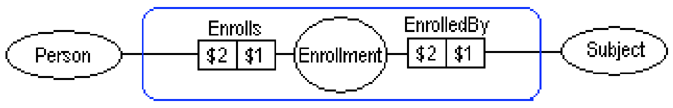
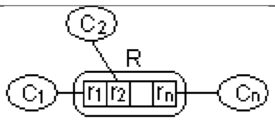


Figure 4.27: Demonstrating the objectified relation "Enrolment" as two binary relations.



$$
\begin{aligned}
R \sqsubseteq\ & [\$1]r_1 \sqcap (\leq 1[\$1]r_1) \sqcap \forall[\$1](r_1 \Rightarrow (\$2 : C_1))\sqcap \\
& [\$1]r_2 \sqcap (\leq 1[\$1]r_2) \sqcap \forall[\$1](r_2 \Rightarrow (\$2 : C_2))\sqcap \\
& \vdots \\
& [\$1]r_n \sqcap (\leq 1[\$1]r_n) \sqcap \forall[\$1](r_n \Rightarrow (\$2 : C_n))
\end{aligned}
$$

(id R $[\$1]r_1, [\$1]r_2, ..., [\$1]r_n$) ***(Rule-29)***

4.2 The Formalization of ORM using $\mathcal{SHOIN}$/OWL

In this section we formalize ORM using $\mathcal{SHOIN}$[HST99], which is the description logic underpinning OWL[4], the standard Ontology Web Language. Compared with $\mathcal{DLR}_{ifd}$, $\mathcal{SHOIN}$ does not support n-ary relations, identification, functional dependencies, and projection of relations, among other things. This implies that several ORM constructs cannot be formalized in $\mathcal{SHOIN}$, and thus are not supported by OWL. These constraints are: predicate uniqueness, external uniqueness, set-comparison constraints (subset, equality, and exclusion) between not contiguous roles, objectification, as well as n-ary relationships.

Without these constraints, mapping ORM into $\mathcal{SHOIN}$ becomes direct based on the mappings presented in the previous section. In other words, mapping ORM into $\mathcal{SHOIN}$/OWL can be seen as a subset of the mappings presented in the previous section. All mapping rules can hold for $\mathcal{SHOIN}$/OWL except {Rule-9,10,16,19,22, and 28}. The syntax of some rules need to be modified. The following table presents these rules according to the $\mathcal{SHOIN}$ syntax, taking into account binary relations only.

To reason about OWL using DL reasoners, OWL statements should be mapped into DIG[TBK$^+$06][5], which is the language that DL reasoners understand. DogmaModeler (as shall be demonstrated in the next section) maps ORM schemes into DIG directly. A functionality to export OWL shall be released in the near future. This functionality is based on table 4.1 (i.e., a subset of ORM).

4.3 Implementation

In this section, we illustrate the implementation of our formalization rules. The formalization is implemented as an extension to the DogmaModeler [Jar05]. DogmaModeler is an ontology modeling tool based on ORM. In DogmaModeler, ORM diagrams are mapped automatically into DIG, which is a description logic interface (XML-based language) that most reasoners (such as Racer, FaCT++, etc) support. DogmaModeler is integrated with the Racer description logic reasoning server which acts as a background reasoning engine. Figure 4.28 shows a screen shot of DogmaModeler. The first window shows an ORM diagram, while the second DIG window shows the *Tell* and *Ask* functionalities. While Tell is to map an ORM diagram into a knowledge base inside Racer, Ask is to reason about this knowledge base. The results of the reasoning about the displayed ORM diagram indicates that the roles “AffiliatedWith” and “Manages“ cannot be satisfied, because the mandatory

[4]OWL can be seen as a friendly XML interface of $\mathcal{SHOIN}$.

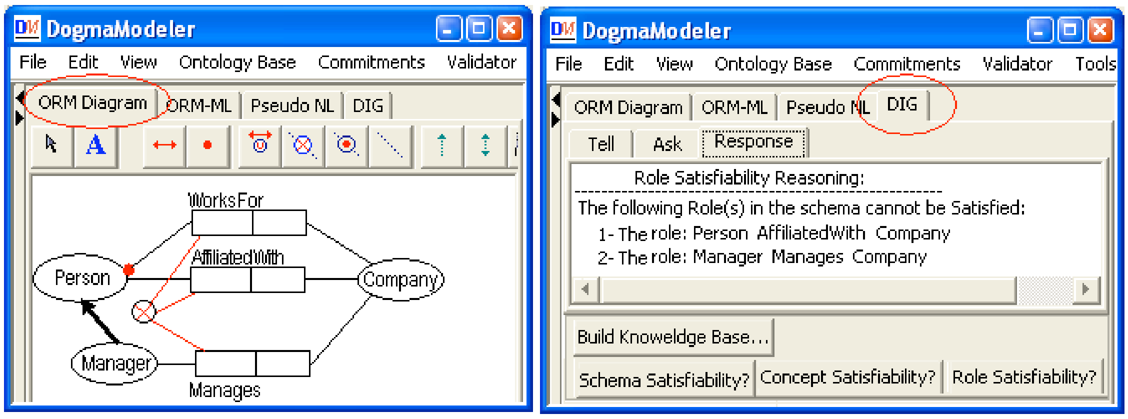
[5]http://dl.kr.org/dig/ (Last visit May 2007)

n-ary relation	$*$	Rule-1
Binary relation	$A \sqsubseteq \forall R.C$	Rule-1'
Unary relation	$A \sqsubseteq \forall R.BOOLEAN$	Rule-2
Role Mandatory	$A \sqsubseteq \exists R.C$	Rule-3
Disjunctive Mandatory	$A \sqsubseteq \exists R_1.C_1 \sqcup .... \sqcup \exists R_n.C_n$	Rule-4
Role Uniqueness	$A \sqsubseteq \leq 1R.C$	Rule-5
Predicate Uniqueness	$*$	Rule-6
External Uniqueness	$*$	Rule-7
Role Frequency	$A \sqsubseteq \exists^{\geq n, \leq m} R.C \sqcup \bot$	Rule-8
Multiple-Role Frequency	$**$	Exception-1
Subtypes	$B \sqsubseteq A$, $A \not\sqsubseteq B$	Rule-9
Total	$A \sqsubseteq A_1 \sqcup A_2 \sqcup ... \sqcup A_n$	Rule-10
Exclusive	$(A_i \sqcap A_j \equiv \bot)$ for each $i \in \{1, .., n_{-1}\}, J \in \{i_{+1}, ..n\}$	Rule-11
String Value	$A \sqsubseteq STRING$, $A \equiv \{x_1, .., x_n\}$	Rule-12
Number Value	$A \sqsubseteq NUMBER$, $A \equiv \{x_1, .., x_n\}$	Rule-13
Role Subset	$\exists R_2.C_2 \sqsubseteq \exists R_1.C_1$	Rule-14
Relation subset	$R_2 \sqsubseteq R_1$	Rule-15
Projection subset	$*$	Rule-16
Role Equality	$\exists R_2.C_2 \equiv \exists R_1.C_1$	Rule-17
Relation Equality	$R_2 \equiv R_1$	Rule-18
Projection Equality	$*$	Rule-19
Role Exclusion	$\exists R_2.C_2 \sqsubseteq \neg\exists R_1.C_1$	Rule-20
Relation Exclusion	$R_2 \sqsubseteq \neg R_1$	Rule-21
Projection Exclusion	$*$	Rule-22
Symmetric	$R \sqsubseteq R^-$	Rule-23
Asymmetric	$R \sqsubseteq \neg R^-$	Rule-24
Antisymmetric	$\top \sqsubseteq (\exists R.Self) \sqcup (\neg\exists R^-.\top)$	Rule-25
Irreflexive	$\top \sqsubseteq \neg\exists R.Self$	Rule-26
Acyclic	$**$	Exception-2
Intransitive	$R \circ R \sqsubseteq \neg R$	Rule-28
Objectification	$*$	Rule-29

Table 4.1: Formalization of ORM using $\mathcal{SHOIN}$; ($*$: Not supported in $\mathcal{SHOIN}$) ($**$: Cannot be formalized in Description logic).

and the exclusion constraints are conflicting each other. DogmaModeler currently implements three types of reasoning services: schema satisfiability, concept satisfiability, and role satisfiability. The other types of reasoning services that are being implemented or are scheduled to be implemented include constraint implications, inference, and subsumption. Please refer to [JE06] for the technical details of DogmaModeler's mapping into DIG.

The main problem we faced during the implementation is that several ORM constraints cannot be mapped into DIG; that is, these constraints were not yet supported by any description logic reasoner. Each formalization rule that could not be implemented is marked by *"Not supported by any DL reasoner yet"* in the previous sections.


DogmaModeler
File Edit View Ontology Base Commitments Validator
ORM Diagram ORM-ML Pseudo NL DIG
WorksFor
AffiliatedWith
Person
Company
Manager
Manages
DogmaModeler
File Edit View Ontology Base Commitments Validator Tools
ORM Diagram ORM-ML Pseudo NL DIG
Tell Ask Response
Build Knoweldge Base...
Schema Satisfiability? Concept Satisfiability? Role Satisfiability?

```
Role Satisfiability Reasoning:
--------------------------------------------------
The following Role(s) in the schema cannot be Satisfied:
   1- The role: Person  AffiliatedWith  Company
   2- The role: Manager  Manages  Company
```

Figure 4.28: DogmaModeler, Ontology modeling using ORM and its automated reasoning.

4.4 Related work

Similar to our work, there have been several efforts to reuse the graphical notation of UML and EER for ontology modeling. Some approaches, such as [BVEL04, KCH$^+$02], considered this to be a visualization issue and did not consider the underpinning semantics. Others (e.g., [SB05, VDSMSJ03]) are motivated only to detect consistency problems in conceptual diagrams. We have found work in formalizing UML in [DBG05], and in [CDGL$^+$98b, BCM$^+$03] for EER. These two formalization efforts have studied the FOL semantics of UML and EER and mapped it into the $\mathcal{DLR}_{ifd}$ description logic. It is also worth noting that the ICOM tool was one of the first tools to enable automated reasoning with conceptual modeling. ICOM [FN00, Ico] supports ontology modeling using a graphical notation that is a mix of the UML and the EER notations. ICOM is fully integrated with the FaCT description logic reasoning server, which acts as a background inference engine.

4.5 Conclusions and future work

In this chapter, we have formalized ORM using both $\mathcal{DLR}_{ifd}$ and $\mathcal{SHOIN}$/OWL, two expressive and wisely used DLs, Our formalization is structured into 29 formalization rules which map all ORM primitives and constraints, except for two complex cases (exception 1 and 2). We have shown which formalization rules can be implemented by current description logic reasoning engines. We have illustrated the implementation of our formalization as an extension to the DogmaModeler. Hence, we have explained how ORM can be used as as a graphical notation for ontology modeling with the reasoning being carried out by a background reasoning engine.

Various issues remain to be addressed. These include extending our formalization to cover more datatypes besides the String, Number, and Boolean types; implementing additional types of reasoning services, specifically constraint implications and inferencing; studying the computational complexity of ORM constraints; and last but not least, is to extend the ORM graphical notation to include some description logical notions, such as the composition, intersection, and union between relations.

Acknowledgment: This research was initiated during my visit to Enrico Franconi at the Free University of Bozen-Bolzano, which was funded by the Knowledge Web project (FP6-507482). I am indebted to Enrico for his very valuable suggestions, contributions, and encouragement. I am also indebted to Sergio Tessaris, Terry Halpin, and Rob Shearer for their valuable comments and feedback on the final version of the chapter. I wish to thank Diego Calvanese, Maurizio Lenzerini, Stijn Heymans, Robert Meersman, Ian Horrocks, Alessandro Artale, Erik Proper, Marijke Keet, and Jeff Pan for their comments and suggestions during this research.

Chapter 5

Conclusion

In this report, we have presented mappings between three different modeling languages—OBO, Structured Ontology Format (SOF), and ORM—and OWL.

In all three cases, the non-OWL modeling syntax offers compelling advantages over OWL's native RDF serialization: OBO provides a simple and intuitive text format or human users, ORM offers a graphical modeling paradigm, and SOF includes a straightforward data model which can be easily used as a library-free ontology API from any popular programming language.

By providing mappings from each of these languages to OWL, we have also made a wide selection of tools available to those working with OWL ontologies: IDEs for OBO syntax, graphical editors for ORM, and scripting libraries for SOF.

Just as importantly, each alternate modeling syntax makes OWL, as well as the large library of OWL-based tools and services, available to a larger audience. OBO is widely used by biologists and there exists a large repository of biological knowledge encoded in OBO. ORM is a mature and well-known language which has been in use in the database community for over thirty years. While SOF is a new syntax, it is designed specifically to appeal to "casual" programmers who would like to work with ontologies using scripting languages. By joining these groups with the OWL community we create a richer ecosystem for the further development of semantic web technologies.

Bibliography

[ABB$^+$00] M. Ashburner, C. A. Ball, J. A. Blake, D. Botstein, H. Butler, J. M. Cherry, A. P. Davis, K. Dolinski, S. S. Dwight, J. T. Eppig, M. A. Harris, D. P. Hill, L. Issel-Tarver, A. Kasarskis, S. Lewis, J. C. Matese, J. E. Richardson, M. Ringwald, G. M. Rubin, and G. Sherlock. Gene Ontolgy: Tool for the Unification of Biology. *Nature Genetics*, 25(1):25–29, 2000.

[BB95] A.T. Berztiss and J.A. Bubenko. A software process model for business reengineering. In *Proceedings of Information Systems Development for Decentralized Organizations (ISDO95), an IFIP 8.1 Working Conference*, pages 184–200, Trondheim Norway, August 1995. Chapman and Hall.

[BCM$^+$03] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.

[BGH99] L. Bird, A. Goodchild, and T.A. Halpin. Object role modelling and xml-schema. In A. Laender, S. Liddle, and V. Storey, editors, *Proc. of the 19th International Conference on Conceptual Modeling*, LLNCS. Springer, 1999.

[BHLT06] Tim Bray, Dave Hollander, Andrew Layman, and Richard Tobin. Namespaces in XML 1.0 (second edition). Technical Report http://www.w3.org/TR/2006/REC-xml-names-20060816/, W3C, August 2006. `http://www.w3.org/TR/2006/REC-xml-names-20060816/`.

[BVEL04] S. Brockmans, R. Volz, A. Eberhart, and P. Loffler. Visual modeling of owl dl ontologies using uml. *Proc. of International Semantic Web Conference (2004)*, pages 198–213, 2004.

[BvHH$^+$04] Sean Bechhofer, Frank van Harmelen, James Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein eds. OWL Web Ontology Language Reference. http://www.w3.org/TR/owl-ref/, Feb 2004.

[BVL03] Sean Bechhofer, Raphael Volz, and Phillip W. Lord. Cooking the semantic web with the OWL API. In Dieter Fensel, Katia P. Sycara, and John Mylopoulos, editors, *International Semantic Web Conference*, volume 2870 of *Lecture Notes in Computer Science*, pages 659–675. Springer, 2003.

[Cas] *CaseTalk website: http://www.casetalk.com/php/ (Visited, October 1, 2006).*

[CDGL98a] D. Calvanese, G. De Giacomo, and M. Lenzerini. On the decidability of query containment under constraints. *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 149–158, 1998.

[CDGL$^+$98b] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. Source integration in data warehousing. In *Proc. of the 9th Int. Workshop on Database and Expert Systems Applications (DEXA'98)*, pages 192–197. IEEE Computer Society Press, 1998.

[CDGL$^+$06] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Data complexity of query answering in description logics. In Patrick Doherty, John Mylopoulos, and Christopher Welty, editors, *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR2006)*, pages 178–218, Menlo Park, California, 2006. AAAI Press.

[CGL01] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Identification constraints and functional dependencies in description logics. In *Proceedings of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)*, pages 155–160, 2001.

[CLN99] D. Calvanese, M. Lenzerini, and D. Nardi. Unifying class-based representation formalisms. *J. of Artificial Intelligence Research*, 11:199–240, 1999.

[DAHtH02] Marlon Dumas, Lachlan Aldred, Mitra Heravizadeh, and Arthur H.M. ter Hofstede. Ontology markup for web forms generation. In *WWW'02 Workshop on Real-World Applications of RDF and the Semantic Web*, 2002.

[DBG05] Diego Calvanese Daniela Berardi and Giuseppe De Giacomo. Reasoning on uml class diagrams. *Artificial Intelligence*, 168(1):70–118, 2005.

[DCv$^+$02] Mike Dean, Dan Connolly, Frank van Harmelen, James Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. OWL web ontology language 1.0 reference, July 2002. `http://www.w3.org/TR/owl-ref/`.

[DJM02] J. Demey, M. Jarrar, and R. Meersman. A conceptual markup language that supports interoperability between business rule modeling systems. In *Proc. of the Tenth International Conference on Cooperative Information Systems (CoopIS 02)*, number 2519 in LNCS, pages 19–35. Springer, 2002.

[DLc] *DL online course by Enrico Franconi, http://www.inf.unibz.it/ franconi/dl/course/ (Visited, October 1, 2006).*

[DNR02] F. M. Donini, D. Nardi, and R. Rosati. Description Logics of Minimal Knowledge and Negation as Failure. *ACM Transactions on Computational Logic*, 3(2):177–225, 2002.

[DRPM06] S. Derriere, A. Richard, and A. Preite-Martinez. An Ontology of Astronomical Object Types for the Virtual Observatory. In *Proc. of the 26th meeting of the IAU: Virtual Observatory in Action: New Science, New Technology, and Next Generation Facilities*, pages 17–18, Prague, Czech Republic, August 21–22 2006.

[dT96] Olga de Troyer. A formalization of the binary object-role model based on logic. *Data and Knowledge Engineering*, 19:1–37, 1996.

[DT05] Casteleyn S. Plessers P. De Troyer, O. Using orm to model web systems. In Terry Halpin and Robert Meersman, editors, *Proceeding of the International Workshop on Object-Role Modeling (ORM'05)*, Agia Napa, Cyprus, 2005. Springer.

[dTM95] Olga de Troyer and Robert Meersman. A logic framework for a semantics of object-oriented data modelling. In M.P. Papazoglou, editor, *Proceedings of 14th International Conference Object-Orientation and Entity-Relationship Modelling*. LNCS 1021, Springer., 238-249 1995.

[FN00] Enrico Franconi and Gary Ng. The i.com tool for intelligent conceptual modelling. In *7th Intl. Workshop on Knowledge Representation meets Databases(KRDB'00)*. Springer Verlag, August 2000.

[GHKS07] B. Cuenca Grau, I. Horrocks, Y. Kazakov, and U. Sattler. Just the Right Amount: Extracting Modules from Ontologies. In *Proc. WWW 2007*, pages 717–726, Banff, AB, Canada, May 8–12 2007.

[GHT06] T. Gardiner, I. Horrocks, and D. Tsarkov. Automated Benchmarking of Description Logic Reasoners. In *Proc. DL 2006*, volume 189 of *CEUR WS Proceedings*, Lake District, UK, May 30–June 1 2006.

[Goo05] J. Goodwin. Experiences of using OWL at the Ordnance Survey. In *Proc. OWL-ED 05*, volume 188 of *CEUR WS Proceedings*, Galway, Ireland, November 11–12 2005.

[GW02] Nicola Guarino and Chris Welty. Evaluating ontological decisions with ontoclean. *Communications of the ACM*, 45(2):6165, 2002.

[GZB06] C. Golbreich, S. Zhang, and O. Bodenreider. The Foundational Model of Anatomy in OWL: Experience and Perspectives. *Journal of Web Semantics*, 4(3):181–195, 2006.

[Hal89] Terry Halpin. *A logical analysis of information systems: static aspects of the data-oriented perspective*. University of Queensland, PhD Thesis, 1989.

[Hal01] Terry Halpin. *Information Modeling and Relational Databases, 3rd edn.* Morgan-Kaufmann, 2001.

[Hal02] Terry Halpin. Join constraints. In Terry Halpin, Keng Siau, and John Krogstie, editors, *Proceedings of the 7th International IFIP WG8.1 Workshop on Evaluation of Modeling Methods in Systems Analysis and Design (EMMSAD'02)*, June 2002.

[Hal04] Terry Halpin. Business rule verbalization. In Halpin T. Liddle S. Mayr H. Doroshenko, A., editor, *Information Systems Technology and its Applications, 3rd International Conference (ISTA'2004)*, pages 39–52, 2004.

[Hal05] Terry Halpin. Objectification. In John Krogstie Terry Halpin, Keng Siau, editor, *Proceddings of the 10th International Workshop on Exploring Modeling Methods in Systems Analysis and Design (EMMSAD05) at CAiSE 2005*, 2005.

[HBN07] M. Horridge, S. Bechhofer, and O. Noppens. Igniting the OWL 1.1 Touch Paper: The OWL API. In *Proc. OWL-ED 2007*, Innsbruck, Austria, June 6–7 2007. To appear.

[HC06] Terry Halpin and Matt Curland. Automated verbalization for orm 2. In Robert Meersman and et al Zahir Tari, editors, *Proceeding of the International Workshop on Object-Role Modeling (ORM'06), OTM 2006 Workshops*. Springer Verlag, 2006.

[HdCD$^+$05] F. W. Hartel, S. de Coronado, R. Dionne, G. Fragoso, and J. Golbeck. Modeling a Description Logic Vocabulary for Cancer Research. *Journal of Biomedical Informatics*, 38(2):114–129, 2005.

[HDG$^+$06] Matthew Horridge, Nick Drummond, John Goodwin, Alan Rector, Robert Stevens, and Hai H Wang. Manchester OWL syntax. In *Proc. of the 2006 OWL: Experiences and Directions Workshop (OWLED 2006)*, 2006.

[HEHM03] T. Halpin, K. Evans, P. Hallock, and W. MacLean. *Database Modeling with Microsoft Visio for Enterprise Architects*. Microsoft Visio, Morgan Kaufmann, San Francisco., 2003,.

[HEPS03] Masahiro Hori, Jérôme Euzenat, and Peter F. Patel-Schneider. OWL web ontology language XML presentation syntax. W3C Note, 11 June 2003. `http://www.w3.org/TR/owl-xmlsyntax/`.

[HKS06] Ian Horrocks, Oliver Kutz, and Ulrike Sattler. The even more irresistible $\mathcal{SROIQ}$. In *Proceeding of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR 2006)*, 2006.

[HM01] V. Haarslev and R. Möller. RACER System Description. In *Proc. IJCAR 2001*, pages 701–706, Siena, Italy, June 18–23 2001.

[HP95] Terry Halpin and Erik Proper. Subtyping and polymorphism in object-role modelling. *Data and Knowledge Engineering*, 15(3):251–281, 1995.

[HPH04] A.I. Bleeker H.A. Proper and S.J.B.A. Hoppenbrouwers. Object-role modelling as a domain. In J. Grundspenkis and M. Kirikova, editors, *Proceedings ofthe Workshop on Evaluating Modeling Methods for Systems Analysis and Design*, volume 3, page 317328, 2004.

[HPSvH03] Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From $\mathcal{SHIQ}$ and RDF to OWL: The making of a web ontology language. *J. of Web Semantics*, 1(1):7–26, 2003.

[HS04] Ian Horrocks and Ulrike Sattler. Decidability of $\mathcal{SHIQ}$ with complex role inclusion axioms. *Artificial Intelligence*, 160(1–2):79–104, December 2004.

[HST99] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Practical reasoning for expressive description logics. In H. Ganzinger, D. McAllester, and A. Voronkov, editors, *Proceedings of the 6th International Conference on Logic for Programming and Automated Reasoning (LPAR'99)*, number 1705 in Lecture Notes in Artificial Intelligence, pages 161–180. Springer-Verlag, 1999.

[Ico] *ICOM website, http://www.inf.unibz.it/ franconi/icom/(Visited, October 1, 2006).*

[Inf] *Infagon website: http://www.mattic.com/Infagon.html (Visited, October 1, 2006).*

[Jar05] Mustafa Jarrar. *Towards Methodological Principles for Ontology Engineering*. PhD thesis, Vrije Universiteit Brussel, Brussels, Belgium, May 2005.

[Jar06a] Mustafa Jarrar. Orm markup language, version 3. Technical report, Vrije Universiteit Brussel, Brussels, Belgium, August 2006.

[Jar06b] Mustafa Jarrar. Towards the notion of gloss, and the adoption of linguistic resources in formal ontology engineering. In *Proceedings of the 15th international conference on World Wide Web (WWW2006)*, pages 497–503, Edinburgh, Scotland., May 2006. ACM Press.

[JDM1] Mustafa Jarrar, Jan Demey, and Robert Meersman. On using conceptual data modeling for ontology engineering. *Journal on Data Semantics (Special issue on Best papers from the ER/ODBASE/COOPIS 2002 Conferences.)*, 2800:185–207, October 1.

[JE06] Mustafa Jarrar and Mohammed Eldammagh. Reasoning on orm using racer. Technical report, Vrije Universiteit Brussel, Brussels, Belgium, August 2006.

[JKD06] Mustafa Jarrar, Maria Keet, and Paolo Dongilli. Multilingual verbalization of orm conceptual models and axiomatized ontologies. Technical report, Vrije Universiteit Brussel, Brussels, Belgium, February 2006.

[JM02] Mustafa Jarrar and Robert Meersman. Formal ontology engineering in the dogma approach. In Robert Meersman and Zahir Tari, editors, *Proceedings of the International Conference on Ontologies, Databases and Applications of Semantics (ODBase 02).*, volume LNCS 2519, pages 1238–1254. Springer Verlag, 2002.

[JM07] Mustafa Jarrar and Robert Meersman. *The DOGMA Approach for Ontology Engineering*, volume 1 of *Advances in Web Semantic, A state-of-the Art Semantic Web Advances in Web Semantics IFIP2.12.*, chapter 3. Springer-sbm, 2007.

[JVM03] Mustafa Jarrar, Ruben Verlinden, and Robert Meersman. Ontology-based customer complaint management. In Mustafa Jarrar and Ann Salaun, editors, *OTM 2003 Workshops, proceedings of the workshop on regulatory ontologies and the modeling of complaint regulations*, volume LNCS 2889, pages 594–606, Catania, Sicily, Italy, 2003. Springer.

[KCH$^+$02] Paul Kogut, Stephen Cranefield, Lewis Hart, Mark Dutra, Kenneth Baclawski, Mieczyslaw Kokar, and Jeffrey Smith. Uml for ontology development. *Knowl. Eng. Rev.*, 17(1):61–64, 2002.

[KFNM04a] H. Knublauch, R. W. Fergerson, N. F. Noy, and M. A. Musen. The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications. In *Proc. ISWC 2004*, pages 229–243, Hiroshima, Japan, November 7–11 2004.

[KFNM04b] Holger Knublauch, Ray W. Fergerson, Natalya Fridman Noy, and Mark A. Musen. The Protégé OWL plugin: An open development environment for semantic web applications. In Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors, *International Semantic Web Conference*, volume 3298 of *Lecture Notes in Computer Science*, pages 229–243. Springer, 2004.

[KPH05a] A. Kalyanpur, B. Parsia, and J. Hendler. A Tool for Working with Web Ontologies. *International Journal on Semantic Web and Information Systems*, 1(1):36–49, 2005.

[KPH05b] Aditya Kalyanpur, Bijan Parsia, and James A. Hendler. A tool for working with web ontologies. *Int. J. Semantic Web Inf. Syst.*, 1(1):36–49, 2005.

[KPSH05] A. Kalyanpur, B. Parsia, E. Sirin, and J. Hendler. Debugging Unsatisfiable Classes in OWL Ontologies. *Journal of Web Semantics*, 3(4):243–366, 2005.

[LAF$^+$05] L. Lacy, G. Aviles, K. Fraser, W. Gerber, A. Mulvehill, and R. Gaskill. Experiences Using OWL in Military Applications. In *Proc. OWL-ED 05*, volume 188 of *CEUR WS Proceedings*, Galway, Ireland, November 11–12 2005.

[MHS07] B. Motik, I. Horrocks, and U. Sattler. Bridging the Gap Between OWL and Relational Databases. In *Proc. WWW 2007*, pages 807–816, Banff, AB, Canada, May 8–12 2007.

[MS06] B. Motik and U. Sattler. A Comparison of Reasoning Techniques for Querying Large Description Logic ABoxes. In *Proc. LPAR 2006*, pages 227–241, Phnom Penh, Cambodia, November 13–17 2006.

[MSH07] B. Motik, R. Shearer, and I. Horrocks. Optimized Reasoning in Description Logics using Hypertableaux. In *Proc. CADE-21*, Bremen, Germany, July 17–20 2007. To appear.

[NOR] *NORMA website http://sourceforge.net/projects/orm , or www.orm.net (Visited, October 1, 2006).*

[Nor99] K. North. Modeling, data semantics, and natural language. *New Architect magazine*, 1999.

[Pip06] Baba Piprani. Using orm-based models as a foundation for a data quality firewall in an advanced generation data warehouse. In Terry Halpin and Robert Meersman, editors, *Proceeding of the International Workshop on Object-Role Modeling (ORM'06)*, volume LNCS. Springer, 2006.

[PSH06a] P. F. Patel-Schneider and I. Horrocks. OWL 1.1 Web Ontology Language Overview. W3C Member Submission, December 19 2006. Available at `http://www.w3.org/Submission/owl11-overview/`.

[PSH06b] Peter F. Patel-Schneider and Ian Horrocks. Owl 1.1 web ontology language overview. W3C Member Submission, 19 December 2006. `http://www.w3.org/Submission/2006/10/`.

[PSHH03] Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks. OWL web ontology language semantics and abstract syntax. W3C Candidate Recommendation, 18 August 2003. Available at `http://www.w3.org/TR/owl-semantics/`.

[PSS] Peter F. Patel-Schneider and Bill Swartout. Description-logic knowledge representation system specification from the KRSS group of the ARPA knowledge sharing effort. `http://www.cs.bell-labs.com/cm/cs/who/pfps/publications/krss-spec.pdf`.

[Qui68] M. R. Quillian. Semantic Memory. In M. Minsky, editor, *Semantic Information Processing*, pages 216–270. MIT Press, Cambridge, MA, USA, 1968.

[RR06] A. Rector and J. Rogers. Ontological and Practical Issues in Using a Description Logic to Represent Medical Concept Systems: Experience from GALEN. In *Reasoning Web, Second International Summer School, Tutorial Lectures*, pages 197–231, Lisbon, Portugal, September 4–8 2006.

[RRL05] A. Ruttenberg, J. Rees, and J. Luciano. Experience Using OWL DL for the Exchange of Biological Pathway Information. In *Proc. OWL-ED 05*, volume 188 of *CEUR WS Proceedings*, Galway, Ireland, November 11–12 2005.

[SB05] J. Simmonds and M.C. Bastarrica. A tool for automatic uml model consistency checking. *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, pages 431–432, 2005.

[SDCS05] A. Sidhu, T. Dillon, E. Chang, and B. Singh Sidhu. Protein Ontology Development using OWL. In *Proc. OWL-ED 05*, volume 188 of *CEUR WS Proceedings*, Galway, Ireland, November 11–12 2005.

[SLL$^+$04] D. Soergel, B. Lauser, A. Liang, F. Fisseha, J. Keizer, and S. Katz. Reengineering Thesauri for New Applications: The AGROVOC Example. *Journal of Digital Information*, 4(4), 2004.

[SP04] E. Sirin and B. Parsia. Pellet: An OWL DL Reasoner. In *Proc. DL 2004*, volume 104 of *CEUR WS Proceedings*, Whistler, BC, Canada, June 6–8 2004.

[TBK$^+$06] A.-Y. Turhan, S. Bechhofer, A. Kaplunova, T. Liebig, M. Luther, R. Moller, O. Noppens, P. Patel-Schneider, B. Suntisrivaraporn, and T. Weithoner. Dig 2.0 towards a flexible interface for description logic reasoners. In B. Cuenca Grau, P. Hitzler, C. Shankey, and E. Wallace, editors, *In Proceedings of the second international workshop OWL: Experiences and Directions*, November 2006.

[tH93] Proper H. van der Weide T ter Hofstede, A. Formal definition of a conceptual language for the description and manipulation of information models. *Information Systems*, 18(7):471–495, October 1993.

[TH06] D. Tsarkov and I. Horrocks. FaCT++ Description Logic Reasoner: System Description. In *Proc. IJCAR 2006*, pages 292–297, Seattle, WA, USA, August 17–20 2006.

[vBtHvdW91] Patrick van Bommel, Arthur H. M. ter Hofstede, and Theo P. van der Weide. Semantics and verification of object-role models. *Information Systems*, 16(5):471–495, 1991.

[VDSMSJ03] R. Van Der Straeten, T. Mens, J. Simmonds, and V. Jonckers. Using description logic to maintain consistency between uml models. *UML*, pages 326–340, 2003.

[vdWtHvB92] Th. P. van der Weide, A. H.M. ter Hofstede, and P. van Bommel. Uniquest: determining the semantics of complex uniqueness constraints. *Comput. J.*, 35(2):148–156, 1992.

[Vis] *VisioModeler download site: http://www.microsoft.com/downloads/ results.aspx?displaylang=en&freeText =VisioModeler (Visited, October 1, 2006).*

[VvB82] G.M.A. Verheijen and J. van Bekkum. Niam: an information analysis method. In T.W. Olle, H.G. Sol, and A.A. Verrijn-Stuart, editors, *Information Systems Design Methodologies: A Comparative Review*, page 537590, Amsterdam, The Netherlands, 1982. x, North-Holland.

[Win90] J.J.V.R. Wintraecken. *The NIAM Information Analysis Method: Theory and Practice.* Kluwer, Deventer, The Netherlands, 1990.

[YAM] YAML specification. `http://www.yaml.org/spec/`.