D2.5.3 Report on Implementation and Optimisation Techniques for Ontology Query Systems

Jeff Z. Pan (UoM)
Enrico Franconi, Sergio Tessaris (FUB), Dmitry Tsarkov, Ian Horrocks (UoM), Giorgos Stoilos, Giorgos Stamou (IT), Holger Wache (VUA), Daniele Turi, Sean Bechhofer, and Lei Li (UoM)

Abstract.
EU-IST Network of Excellence (NoE) IST-2004-507482 KWEB
Deliverable D2.5.3 (WP2.5)

Keyword list: description logics, ontology language, query language, RDF, OWL DL, OWL-E

In D2.5.2 "Report on Query Language Design and Standardisation", we have investigated query answering of conjunctive queries in the OWL-QL framework, including treating RDF triples as individual axioms of an ontology and extending the OWL-QL framework on queries related to datatype expressions over OWL-E ontologies. In this deliverable, we look at implementation and optimisation aspects of ontology query systems. We believe this work would be a very useful step towards proposing standard Semantic Web query languages.

Document Identifier	KWEB/2004/D2.5.3/v1.0
Project	KWEB EU-IST-2004-507482
Version	v1.0
Date	June 25, 2005
State	draft
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

Changes

Version	Date	Author(s)	Changes
0.1	12.04.05	Jeff Z. Pan	creation
0.2	09.05.05	Jeff Z. Pan	Adding the “FaCT-DG” chapter
0.3	12.05.05	Dmitry Tsarkov and Ian Horrocks	Adding the “FaCT++” chapter
0.4	16.05.05	Enrico Franconi and Sergio Tessaris	Adding the “Query Formulation” chapter
0.45	16.05.05	Stefano David, Enrico Franconi, and Sergio Tessaris	Adding the “Query Engines” Section in Chapter 2
0.5	16.05.05	Giorgos Stoilos and Giorgos Stamou	Adding the “Implementing an f-$\mathcal{SI}$ Reasoner” chapter
0.6	16.05.05	Ian Horrocks, Daniele Turi and Sean Bechhofer, Lei Li	Adding the “Instance Store” chapter

Executive Summary

In D2.5.2 “Report on Query Language Design and Standardisation”, we have investigated query answering of conjunctive queries in the OWL-QL framework, including treating RDF triples as individual axioms of an ontology and extending the OWL-QL framework on queries related to datatype expressions over OWL-E ontologies. In this deliverable, we look at implementation and optimisation aspects of ontology query systems. We believe this work would be a very useful step towards proposing standard Semantic Web query languages.

Contents

Chapter 1

Introduction

In D2.5.2 "Report on Query Language Design and Standardisation", from the theoretical aspect, we have investigated query answering of conjunctive queries in the OWL-QL framework, including treating RDF triples as individual axioms of an ontology and extending the OWL-QL framework on queries related to datatype expressions over OWL-E ontologies [Pan04]. In this deliverable, we look at implementation and optimisation aspects of ontology query systems. We believe this work would be a very useful step towards proposing standard Semantic Web query languages.

It is worth noting that, by query systems, we mean both TBox query and retrieval query systems. Modern Description Logics reasoners, such as FaCT [Hor99], RACER [HM01d], DLP [PS99], Cerebra[1] and Pellet, provide some basic querying support (also called *TBox* reasoning), including concept satisfiability, subsumption and classification queries. Some, such as RACER, Cerebra and Pellet, support instance checking and retrieval (also called *ABox* reasoning). Based on basic reasoning tasks like retrieval, one can provide more expressive (conjunctive) queries.

The deliverable is structured as follows. First of all, we provide a short survey of many existing Semantic Web query engines (Chapter 2). Our aim is to present a big picture about what functionalities these systems provide and how well they conform to the Semantic Web standards.

The second part of the deliverable (Chapter 3 to 5) presents some newly developed DL reasoning engines or algorithms, which provide some basic querying support. FaCT++ (Chapter 3) is a new generation of the FaCT reasoner, which provides some new optimisations for efficient reasoning support for OWL Lite ontologies. FaCT-DG (Chapter 4) extends the FaCT reasoner with support of datatype groups and uses a flexible architecture of supporting customised datatypes and datatype predicates. An algorithm for f-$\mathcal{SI}$-concept satisfiability w.r.t. fuzzy knowledge bases is presented in Chapter 5.

The third part of the deliverable (Chapter 6 to 8) discusses several aspects on imple-

[1]`http://www.networkinference.com`

mentations of Semantic Web query systems. Chapter 6 describes the principles of the design and development of intelligent query interface systems. Chapter 7 elaborates implementation issues on approximating terminological queries.

While existing techniques for *TBox* reasoning (i.e., reasoning about the concepts in an ontology) seem able to cope with real world ontologies [Hor98b, HM01b], it is not clear if existing techniques for *ABox* reasoning (i.e., reasoning about the individuals in an ontology) will be able to cope with realistic sets of instance data. This difficulty arises not so much from the computational complexity of *ABox* reasoning, but from the fact that the number of individuals (e.g., annotations) might be extremely large.

To address this problem, Chapter 8 discusses how to combine Description Logic TBox reasoning and databases to facilitate efficient query answering of retrieval queries over extremely large numbers of individuals.

Chapter 9 concludes the deliverable.

Chapter 2

Existing Semantic Web Query Engines

In this chapter we present an evaluation of the most important tools for accessing and manipulating RDF(S) (see[MM04]) and OWL (see [MvH04]) ontologies. Our goal was to understand the reasoning abilities of the tools with respect to the semantic definitions of the problems at hand. We have tested the following systems: Jena, Sesame, RdfSuite, Triple, RdfStore, 4Suite, RdfDB. The main results so far are: (1) RDF(S) is unable to express basic representation needs typical in simple ontologies; (2) few tools are unable to correctly handle RDF(S) ontologies; (3) as of today, no semantic web tool is able to correctly handle even OWL-Lite ontologies, nor they plan to include it; and (4) the only tool able to correctly handle OWL-Lite ontologies are the description logic inference engine Racer and the Manchester OWL-QL Server [GH04], which extends Racer to support non-distinguished variables. The conclusion is that the current technology is unable to handle correctly the W3C standard for representing ontologies.

2.1 Methodology of work

Roughly speaking the evaluation was performed by setting up some simple but tricky ontology models, express them in the W3C standard languages RDF(S) and OWL and evaluate the outcome of some of the most used Semantic Web query tools.

The work has been divided into three subsequent steps: the first and second step are devoted to the acquirement of the technical background and logical understanding of the involved technologies, whereas the third and final step is the practical one, in which simple ontology models are set up and tested against the tools. These steps can be identified as:

1. the study of *knowledge representation*
2. the study of the W3C *ontology languages*
3. the verification of the correctness of the *query tools*

2.1.1 Knowledge representation

The starting point is the study of the Description Logics formalism, of the reasoning processes and of the inference mechanism, to become acquainted with the theoretical background that is necessary to deal with ontologies and reasoning processes. In particular, the key point is represented by two properties of inference: soundness and completeness. Soundness means that every information added to the knowledge base trough the inference process is correct (i.e., the new information is neither contradictory nor wrong); whereas completeness means that the system is able to obtain all the possible additional deducible information.

2.1.2 Ontology languages

We then different layers of expressiveness, which allow for different expressive needs. The basic layer is represented by RDF(S), while the following are OWL/Lite, OWL/DL and OWL/Full. Beside these standard OWL sublanguages, there is also a proposal for OWL/Lite-[1] (an implementation of DLP[GHVD03]), proposed by the SDK WSMO Working Group. However, since this work is intended to deal only with tools and language that support W3C Standards, OWL/Lite- has not been taken into account. Moreover, OWL/Full was also discarded, because it does not guarantee to have the capability to complete the inference processes in a finite time, nor to be sound and complete.

2.1.3 Query tools

Finally, the last effort is about the tools: we had to decide

- which of them to use
- how to evaluate them

The former question was answered after the consideration that giving an answer to a query requires the capability to take advantage from inference processes, so we decided to examine only those tools that implement some kind of inference engine. The tools used during the development of this paper are therefore: Jena, RDFSuite, Sesame, Triple and Racer. Racer is not a Semantic Web Tool, but a reasoner developed inside the Description Logics community, and it is used in this work as a comparison. The latter question found its answer in the consideration that the best way to verify the correctness of the outcome is to create easy but tricky ontology models, whose domains lie on the border of the layers identified above, and check whether the results obtained by the tools coincide with the result we expect from the models.

[1]http://www.wsmo.org/2004/d20/d20.2/v0.2/20041123/

2.2 The test cases

The idea of our work was to create a set of models, starting from a very simple ontology encoded in RDF(S), developing it by adding properties and classes to it (this model will be referred to as *"Course"*), in order to incrementally explore the whole RDF(S) and OWL capabilities. However, we soon realized that it was not possible to add much complexity to our original model without having to switch to OWL. Hence we created a second model (*"British"*), with the purpose to shows the limits of RDF(S). The next model (*"Teacher"*) was set up to show that there are simple ontologies that can not be expressed by the constructs of OWL/Lite-.
After developing our first model in OWL/Lite (*"Friends"*), we stopped investigating the further layers, since we realized that at that point of expressiveness there is no Semantic Web tool that can deal with ontologies of this complexity. In terms of expressiveness, our four simple ontology models are placed among (and should represent a good coverage of) all the layers of RDF(S) and OWL: *Course* is entirely in RDF(S), *British* between RDF(S) and OWL/Lite, *Teacher* in OWL/Lite, just outside OWL/Lite- and *Friends* is in an encoding in OWL/Lite of an OWL/DL ontology. This is possible since these two sublanguages have the same expressive power. The rest of this section presents the models, that are also presented graphically, along with the query we ask the ontology and the expected result.

To allow for a better understanding, the knowledge bases of the models are written in Notation3 (see [BL01]); moreover, the base namespaces of the ontologies are omitted.

The Course Model

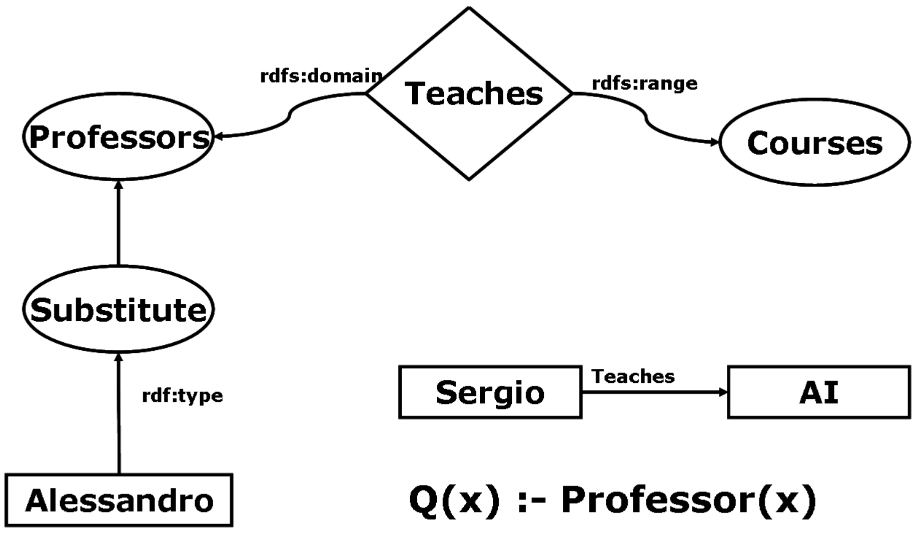


Figure 2.1: The Course model.

As it can be seen in Figure 2.1, our first model is a very simple ontology about professors who teach courses. From this model we want an answer to the following query:

$$Q(X) := Professor(X)$$

that is, we want to know whether there are professors in the model and who they are.

This model has the purpose to force some basic constraint of RDF(S), namely *rdfs:subClassOf* and both *rdfs:range* and *rdfs:domain*. The former can be seen as a verification of the taxonomy in the domain, while the latter is more complicated, in the sense that it forces the two basic constraints of a property: its domain (e.g., the classes that a Property can have as subject in a RDFS statement) and its range (e.g., the object of the RDFS statement). In other words, domain and range constrain subject and object of the property to be instances of a certain class. Moreover, if multiple classes are given as domain (range) of the same property, the subject (object) of the statement is an instance of all these classes.
Clearly, we can easily infer from the model that we have an answer to the query, which is the set {*Alessandro, Sergio*}. Alessandro is the result of the inference over the taxonomy of the domain and Sergio comes from the constraint on the domain of the property Teaches.
Note that besides this answer, there is also an additional information deduced from the model (AI is of type Course) and is due to the constraint on the range of the property Teaches.

The Teacher Model

Despite its simplicity, this model has a need for expressiveness that places it outside the extent of OWL/Lite-. The concept we want to express is “a teacher is someone who teaches a course”. Figure 2.2 shows how this concept can be drawn as an OWL/Lite graph. We also know that Sergio teaches Artificial Intelligence, which is a Course. This ontology should return an answer to this query:

$$Q(X) := Teacher(X)$$

Like in the previous model, we wanted to force a constraint on a property. The difference with RDF(S) is to be found in the expression of domain/range constraint: OWL allows for having multiple ranges and domains by means of a Restriction on a Property. In this case, Teacher is a Class, equivalent to another Class, which is a restriction on the property teaches to those values that are instances of class Course. This restriction, along with the knowledge of Sergio teaching a course (AI), is all we need to deduce that our answer set is not empty but contains an element, {*Sergio*}.

The Friends Model

This is perhaps the most trickier model we set up. It is about friends, lovers, Managers and Employees and consists of only four individuals, two classes and three properties,

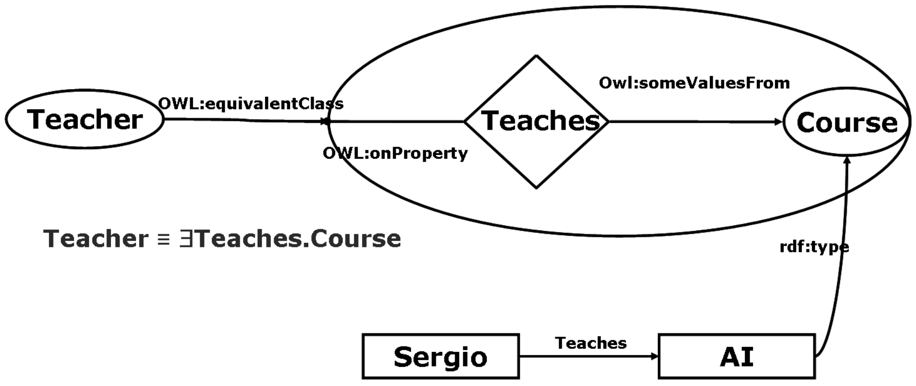


Figure 2.2: The Teacher model.

how Figure 2.3 shows; we also consider the domain of our ontology to be composed only by workers (i.e., $worker \sqsupseteq \top$).

We focus on an additional information, which is also included in the ontology; we keep it distinct from the previous, since it is the key point in this ontology:

All workers are either Managers or Employee.

But why is this information so important? First of all because it partitions the domain in two parts: managers or employees, no other possibility. Moreover, this model can be easily express in OWL/DL, where the statement above can be written as, for example,

```
<owl:Class rdf:about="#WORKER">
 <owl:equivalentClass>
  <owl:Class>
   <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#MANAGER"/>
    <owl:Class rdf:about="#EMPLOYEE"/>
   </owl:unionOf>
  </owl:Class>
 </owl:equivalentClass>
</owl:Class>
```

A second method to write the above statement would be to use the negation (e.g., $Manager \equiv \neg Employee$, that corresponds to the OWL construct `owl:complementOf`. Both this two method need OWL/DL, since both `owl:unionOf` and `owl:complementOf` are not allowed in OWL/Lite. However, it is not so immediate that the same information can also be expressed in OWL/Lite, by adding a new property (e.g., hasBoss) and exploiting OWL/Lite limited cardinality restriction.

We can say that a Manager is anyone whose hasBoss Property has an `owl:minCardinality` of 1, whereas Employee is anyone who has an `owl:maxCardinality` of 0 on the same Property.

Given this model we want to have an answer the following query:

$$Q(X) := hasFriend(X,Y) \land Manager(Y) \land loves(Y,Z) \land Employee(Z)$$

That is, we are looking for someone who has a manager as friend and this manager loves an employee. Apparently, this query seems to have no answer, due to the lack on knowledge about Andrea. However, we can take advantage from the so called *reasoning by case* to find an answer. Hence consider Andrea, who might be (according to our model) either Employee or Manager.
In the former case (i.e., Andrea is an Employee), we can deduce that Paul has a friend (Simon), who is a manager, and Simon loves an Employee (Andrea). This gives a partial answer to the proposed query.
Also in the latter case (i.e., Andrea is a Manager), Paul has a friend (Andrea) who is a Manager, and Andrea loves an Employee (Caroline).
Since there is no other possible value for Andrea and in both cases we found in Paul an answer to our query, we can state that the query has an answer: $\{Paul\}$

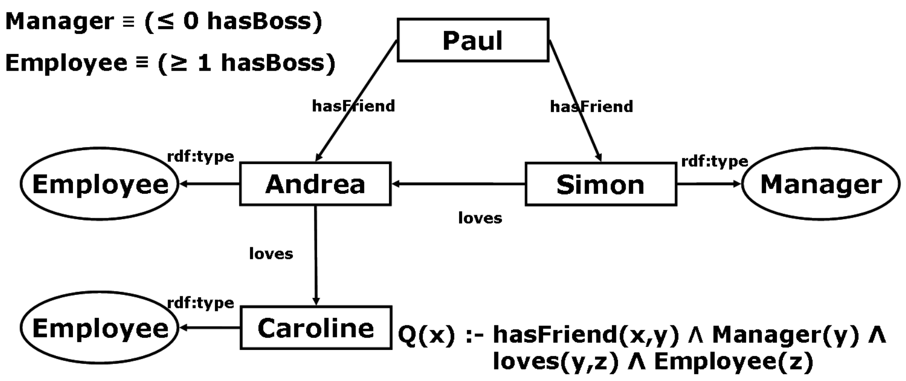


Figure 2.3: The Friends model.

The British Model

We have now reached an expressiveness level at which all Semantic Web tools fail, so let us make a step back and present an example that shows how RDF(S) may unveil its inability to capture simple knowledge representation needs. Figure 2.4 shows such an example. The idea of this ontology model is to express the following statements:

- Italian people live in Italian cities

- British people live in British cities
- John is British
- John lives in London

with the intermediate deduction that John lives in a British city, we conclude that London is a British city. Our query is actually:

$$Q(X) := GB(X)$$

Although we expressed this model in two RDF(S) versions, this query can not be answered. The first version is, like in Figure 2.4, a hierarchy on the property hasResidence, while the second a hierarchy on the classes (i.e., hasResidence has as domain the class Citizen, which is subclassed by Italian and British and as range the class Country, subclassed by Italy and GB). Our ontology has an answer to the above query, $\{London\}$, but RDF(S) can not derive such an information. Hence we provided an OWL/Lite version of this model, since OWL allows more constructs for disambiguating knowledge.

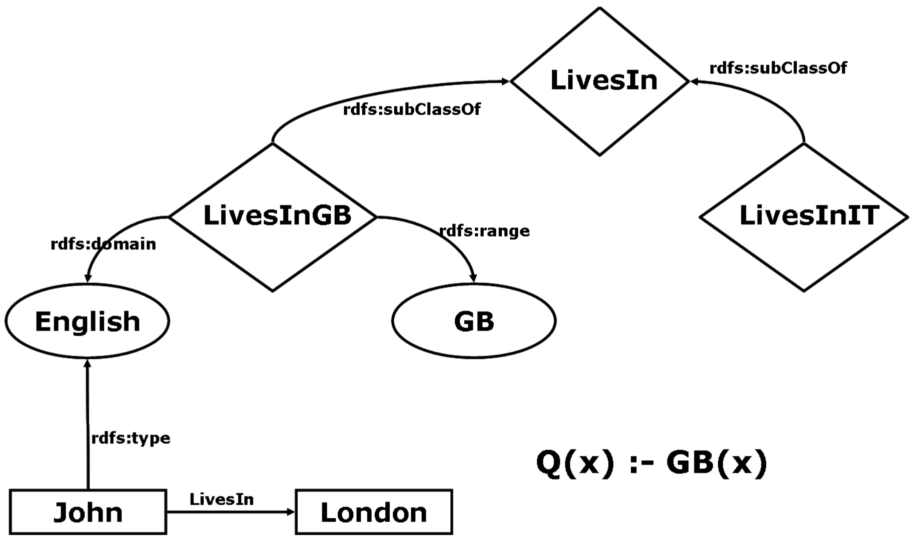


Figure 2.4: The British model.

2.3 Evaluation results

Table 2.1 summarizes the result returned by the tools we considered when they are fed with our models. There are two different aspects to consider, in the discussion of the results. On the one side, in RDF(S), almost all tool can correctly answer to the queries, on the other side, in OWL, there is still a lot of work to do.

Table of results					
	Jena	RDFSuite	Sesame	Triple	Racer
Courses RDF(S)	ok	no	ok	part	ok
Teacher OWL/Lite	ok	–	–	–	ok
Friends OWL/Lite	er	–	–	–	ok
British OWL/Lite	ok	–	–	–	ok

ok:	test passed	**no**:	test not passed
part:	test partially passed	**er**:	test passed with the help of an external reasoner
–:	model not tested with the tool		

Table 2.1: The outcomes of the tools

However, all of these tools are under development, thanks also to the request from the Semantic Web community for tools that support OWL. In particular, below is a brief presentation of the outcomes of each tool. Note that we only consider the tools only for the part of inference implementation, which is the main point of this work, and we disregarded their other useful feature (e.g., database storage, N3, OWL and RDF(S) input/output, and so on).

Jena (see [CDD+04], `http://jena.sorceforge.net`) is perhaps the most complete tool (w.r.t. inference) available at the moment. It can correctly deal with (and infer correct results from) all our ontology models, except from the friends model. However, in that case, its API can rely on an external reasoner (such as Racer) by means of a DIG interface: it sends the whole information to the reasoner, which will send back to the Jena API the results of the inference.

RDFSuite (see [ACK+01], `http://athena.ics.forth.gr:9090/RDF`) is an application composed by three different tools, RSSDB, VRP and RQL. We focused on RSSDB and VRP, but unfortunately, we found some trouble in using them: RSSDB produced some errors during the set up of the postgres database, while VRP did not find the correct answers. In both cases, the results were notified to the authors; we are waiting for some feedback from them, in order to recover from the errors and complete the tests. RDFSuite has support only for RDF(S), so the OWL models were not tested.

Sesame (see [BKvH02], `http://www.openrdf.org`) correctly found the results in the RDF(S) model. However, it only has a limited support for OWL, in particular it currently implements a set of rules that cover the DLP subset of OWL and therefore it was not tested against the OWL models. Support for a more expressive fragment of OWL is already planned, but yet an open issue.

Triple (see [SD02], `http://triple.semanticweb.org`) did not infer all the results we expected, even in the RDF(S) model. In fact, Triple has only a simple set of hardcoded rules, that allows only for inference over RDF(S) taxonomies, but not over domain/range constraints of Properties. It has also support for DAML/Oil, but not for OWL, so it was not tested with the OWL models. In more details, it can only infer the result that Alessandro is a Professor in the Course model.

Racer (see [HM03], `http://www.cs.concordia.ca/~haarslev/racer/`) is a reasoning agent that supports inference over different ontologies, which can be expressed in different languages, among them DAML+Oil and OWL. It was used also with Jena and with Oiled (the ontology editor we used for creating our models), to check the correctness of our ontologies and that the queries could have an answer. It was also used as a standalone tool, by means of RICE (a graphical interface for Racer), and it is actually the only tool, among the ones we tested, that offers inference support for OWL/DL ontologies.

2.4 Conclusion and further works

After the development of this work, it is possible to identify several arguments that are of interest and might be object of discussion. First of all, during the creation of the RDF(S) models, a side-effect emerged, that involved the semantic of RDF(S) and in particular the reasoning process over the domain and range of Properties. This has been briefly described in section 2 and explained here. For example, consider the following snippet of an RDF(S) ontology:
P rdfs:range B
  C rdfs:subClassOf B
  D rdfs:subClassOf B
where P is a Property and B, C, D are Classes.
The additional statement *aClass P z*, together with the above KB, results in the inability of RDF(S) to derive whether z is of type C or D: it is only possible to say that z is of type B. There is no possibility in RDF(S) to further specify the knowledge to disambiguate the statement. This limit is overcome by OWL, whose semantics is far more expressive than the one of RDF(S) and manages in a better way the hierarchies of types in the domain of an ontology, thanks to the capability to specify restrictions on Properties.
The next point is that, even tough RDF(S) is not so expressive (and therefore should be fairly simple to be implemented), not every tool is able to handle ontologies expressed in this language: it is the case of Triple, whose hard coded rules only allow inference on types (Resources) but not on domain/range constraints of Properties, and of RDFSuite.
Perhaps the most relevant aspect revealed by this work applies to the actual software applications that are not able to correctly deal with all kind of OWL ontologies, not even those expressed in OWL/Lite. In details, no tool can take advantage of the reasoning by

case approach, which is probably the most powerful aspects of the DLs formalism (the open world semantics); the example of the *Friends* model is the most significant one. However, there is an exception: it is the case of RACER, a tool developed inside the Description Logics community, which is actually the only software application that can handle all OWL ontologies (except those written in OWL/Full). Some tool, like Jena can rely on RACER, by means of an API that allows to connect to external reasoners, pass an ontology to it, use it for computing inference over the ontology and retrieve the results.
An additional issue in the development of these kind of tools might be represented by the use of non-standard ontology languages (e.g., like OWL-Lite-), that might lead to a waste of efforts of the Semantic Web community, if such language would not be recognized as standard.
A final remark is about query languages for ontologies. At the present time, there are different QL (RQL, RDQL, SeRQL and so on). They mainly differ in their syntax, because all of them work in the same way: they iterate over a RDF or OWL model and search for pattern in the triple stored in memory (or in the underlying database). A standard query language would avoid wasting of efforts in the research and development of software that implements these query languages.
The very last subject is about proposal and future work related to the present one. Important working and research areas to improve the development of Semantic Web Tools might include:

- definition for a new standard query languages, improve the actual implementation of the OWL inference engines, by adding new set of rules that allow to deduce all possible implicit knowledge of a given ontology.
- investigate the best way to implement a reasoning by case approach, to take full advantage from Description Logics formalism.
- further test (e.g., with RDFSuite of with new software applications that will be released
- the creation of a whole suite of test cases to be proposed as standard for testing a Semantic Web tool

Chapter 3

FaCT++: An Efficient OWL Lite Reasoner

Most modern DL systems are based on tableaux algorithms. Such algorithms were first introduced by Schmidt-Schauß and Smolka [SSS91], and subsequently extended to deal with ever more expressive logics [BCM$^+$03]. Many systems now implement the $\mathcal{SHIQ}$ DL, a tableaux algorithm for which was first presented in [HST99]; this logic is very expressive, and corresponds closely to the OWL ontology language. In spite of the high worst case complexity of the satisfiability/subsumption problem for this logic (ExpTime-complete), highly optimised implementations have been shown to work well in many realistic (ontology) applications [Hor98b].

Optimisation is crucial to the viability of tableaux based systems: in experiments using both artificial test data and application ontologies, (relatively) unoptimised systems performed very badly, often being (at least) several orders of magnitude slower than optimised systems; in many cases, hours of processing time (in some cases even hundreds of hours) proved insufficient for unoptimised systems to solve problems that took only a few milliseconds for an optimised system [Mas99, HPS98]. Modern systems typically employ a wide range of optimisations, including (at least) those described in [BFH$^+$94, HPS99].

Tableaux algorithms try to construct a graph (usually a tree) representation of a model of a concept, the structure of which is determined by syntactic decomposition of the concept. Most implementations employ a space saving optimisation known as the *trace technique* that uses a top-down construction requiring (for PSpace logics) only polynomial space in order to delineate a tree structure that may be exponential in size (with respect to the size of the input concept). For the ExpTime logics implemented in modern systems, however, guaranteeing polynomial space usage is no longer an option. Moreover, for logics that support inverse roles (such as $\mathcal{SHIQ}$), a strictly top down approach is no longer possible as constraints may be propagated both "up" and "down" the edges in the tree.

We describe an alternative architecture for tableaux implementations that uses a (set of) queue(s) instead of (an adaption of) the standard top-down approach. This architecture, which we have implemented in our new FaCT++ system, has a number of advantages when compared to the top-down approach. Firstly, it is applicable to a much wider range of logics, including the expressive logics implemented in modern systems, because it makes no assumptions about the structure of the graph (in particular, whether tree shaped or not), or the order in which the graph will be constructed. Secondly, it allows for the use of more powerful heuristics that try to improve typical case performance by varying the global order in which different syntactic structures are decomposed; in a top-down construction, such heuristics can only operate on a local region of the graph—typically a single vertex.

3.1 Architecture

As discussed above, many implementations use a top-down expansion based on the trace technique. The idea of the top-down expansion is to apply the $\exists$-rule with the lowest priority (i.e., only apply this rule when no other rule is applicable); the added refinement of the trace technique is to discard fully expanded sub-trees, so that only a single “trace” (i.e., a branch of the tree) is kept in memory at any one time.

This technique has the advantage of being very simple and easy to implement—a procedure that exhaustively expands a node label can be applied to the current node and then, recursively, to each of its successors. It does, however, have some serious drawbacks. In the first place, for logics with inverse roles, the top-down method simply breaks down as it relies on the fact that rules only ever add concepts to the label of the node to which they are applied or to the label of one of its successor nodes. The result is that, once the rules have been exhaustively applied to a given node label, no further expansion of that label will be possible. In the presence of inverse roles, expansion rules may also add concepts to the labels of predecessor nodes, which could then require further expansion. Moreover, discarding fully expanded sub-trees may no longer be possible, as the expansion of a concept added to the label of a predecessor may cause concepts to be added to the label of a sibling node that had previously been fully expanded.

In the second place, the top down method forces non-deterministic rules to be applied with a higher priority than generating rules. As the size of the search space caused by non-deterministic rule expansions is, in practice, by far the most serious problem for tableaux based systems [Hor97], it may be advantageous to apply non-deterministic rules with the lowest priority [GS96]. In fact, top-down implementations typically apply non-deterministic rules with a priority that is lower than that of all of the other rules *except* the generating rules [HPS99].

ToDo List Architecture The FaCT++ system was designed with the intention of implementing DLs that include inverse roles, and of investigating new optimisation techniques, including new ordering heuristics. Currently, FaCT++ implements $\mathcal{SHIF}$, a slightly less expressive variant of $\mathcal{SHIQ}$ where the values in atleast and atmost restrictions can only be zero or one.[1]

Instead of the top-down approach, FaCT++ uses a *ToDo list* to control the application of the expansion rules. The basic idea behind this approach is that rules may become applicable whenever a concept is added to a node label. When this happens, a note of the node/concept pair is added to the ToDo list. The ToDo list sorts all entries according to some order, and gives access to the "first" element in the list.

A given tableaux algorithm takes an entry from the ToDo list and processes it according to the expansion rule(s) relevant to the entry (if any). During the expansion process, new concepts may be added to node labels, and hence entries may be added to the ToDo list. The process continues until either a clash occurs or the ToDo list become empty.

In FaCT++ we implement the ToDo list architecture as a set of queues (FIFO buffers). It is possible to set a priority for each rule type (e.g., $\sqcap$ and $\exists$), and a separate queue is implemented for each unique priority. Whenever the expansion algorithm asks for a new entry, it is taken from the non-empty queue with the highest priority, and the algorithm terminates when all the queues are empty. This means that if the $\exists$-rule has a low priority (say 0), and all other rules have the same priority (say 1), then the expansion will be (modulo inverse roles) top-down and breadth first; if stacks (LIFO buffers) were used instead of queues with the same priorities, then the expansion would simulate the standard top-down method.

3.2 Heuristics in Implementation

When implementing reasoning algorithms, heuristics can be used to try to find a "good" order in which to apply inference rules (we will call these *rule-ordering* heuristics) and, for non-deterministic rules, the order in which to explore the different expansion choices offered by rule applications (we will call these *expansion-ordering* heuristics). The aim is to choose an order that leads rapidly to the discovery of a model (in case the input is satisfiable) or to a proof that no model exists (in case the input is unsatisfiable). The usual technique is to compute a weighting for each available option, and to choose the option with the highest (or lowest) weight. Much of the "art" in devising useful heuristics is in finding a suitable compromise between the cost of computing the weightings and their accuracy in predicting good orderings.

Such heuristics can be very effective in improving the performance of propositional satisfiability (SAT) reasoners [Fre95], but finding useful heuristics for description and modal logics has proved to be more difficult. Choosing a good heuristic, or at least not choosing a bad one, is very important: an inappropriate heuristic may not simply fail to improve performance, it may seriously degrade it. Even more problematical is, given a range of possible heuristics, choosing the best one to use for a given (type of) problem.

So far, the heuristics tried with DL reasoners have mainly been adaptions of those

[1]$\mathcal{SHIf}$ corresponds to the OWL-Lite ontology language [HPSvH03].

already developed for SAT reasoners, such as the well known MOMS heuristic [Fre95] and Jeroslow and Wang's weighted occurrences heuristic [JW90]. These proved to be largely ineffective, and even to degrade performance due to an adverse interaction with backjumping [BCM+03]. An alternative heuristic, first presented in [Hor97], tries to maximise the effect of dependency directed backtracking (backjumping) by preferentially choosing expansions that introduce concept with "old" dependencies. Even this heuristic, however, has relatively little effect on performance with realistic problems, e.g., problems encountered when reasoning with application ontologies.

We conjecture that the standard top-down architecture has contributed to the difficulty in finding useful heuristics as it rules out many possible choices of rule-ordering; in particular, the top-down technique may require generating rules to be applied with a low priority, and so lead to non-deterministic rules being applied before deterministic generating rules. In contrast, the ToDo list architecture gives a much wider range of possible rule orderings, and so has allowed us to investigate a range of new rule-ordering heuristics, in particular heuristics that give non-deterministic rules the lowest priority.

Another factor that has contributed to the weakness of SAT derived heuristics is that they treat concepts as though they were atoms. This is obviously appropriate in the case of propositional satisfiability, but not in the case of concept satisfiability where sub-concepts may have a complex structure. We have also investigated expansion-ordering heuristics that take into account this structure, in particular a concept's size, maximum quantifier depth, and frequency of usage in the knowledge base.

Implementation in FaCT++ The FaCT++ reasoner uses the standard backtracking search technique to explore the different possible expansions offered by non-deterministic rules (such as the $\sqcup$-rule). Before applying a non-deterministic rule, the current state is saved, and when backtracking, the state is restored before re-applying the same rule (with a different expansion choice). When inverse roles are supported, it is possible for a sequence of deterministic rule applications to propagate changes throughout the graph, and it may, therefore, be necessary to save and restore the whole graph structure (in addition to other data structures such as the ToDo list). FaCT++ trys to minimise the potentially high cost of these operations by lazily saving the graph, (i.e., saving parts of the graph only as necessitated by the expansion), but the cost of saving the state still makes it expensive to apply a non-deterministic rule, even if the state is never restored during backtracking.

As discussed in Section 3.1, FaCT++ uses a ToDo list architecture with separate queues for each priority level. Different rule-ordering heuristics can, therefore, be tried simply by varying the priorities assigned to different rule types. Low priorities are typically given to generating and non-deterministic rules, but the ToDo list architecture allows different priority ordering of these rule types; in contrast, the top-down architecture forces a lower priority to be given to generating rules.

FaCT++ also includes a range of different expansion-ordering heuristics that can be used to choose the order in which to explore the different expansion choices offered by the non-deterministic $\sqcup$-rule. This ordering can be on the basis of the size, maximum quantifier depth, or frequency of usage of each of the concepts in the disjunction, and the order can be either ascending (smallest size, minimum depth and lowest frequency first) or descending. In order to avoid the cost of repeatedly computing such values, FaCT++ gathers all the relevant statistics for each concept as the knowledge base is loaded, and caches them for later use.

3.3 Empirical Analysis

In order to evaluate the usefulness of the heuristics implemented in FaCT++, we have carried out an empirical analysis using both real-life ontologies and artificial tests from the DL'98 test suite [HPS98].

Ontologies can vary widely in terms of size and complexity (e.g., structure of concepts, and types of axiom used). We used three ontologies with different characteristics in order to see how the heuristics would perform in each case:

WineFood A sample ontology that makes up part of the OWL test suit[2] [CDR04]; it is small, but has a complex structure and includes 150 GCIs.

DOLCE A foundational (top-level) ontology, developed in the WonderWeb project [GGM$^+$02]; it is of medium size and medium complexity.

GALEN The anatomical part of the well-known medical terminology ontology [RRS$^+$01]; it is large (4,000 concepts) and has a relatively simple structure, but includes over 400 GCIs.

FaCT++ separates the classification process into satisfiability testing (SAT) and subsumption testing (SUB) phases; the results from the SAT phase are cached and used to speed up subsequent tests via a standard "model-merging" optimisation [HPS99]. FaCT++ allows different heuristics to be used in the two phases of the process; this is because the tests have different characteristics: in the SAT phase, nearly all of the tests are satisfiable (ontologies typically do not give names to unsatisfiable concepts), while in the SUB phase, up to one in four of the tests are unsatisfiable. We measured the time (in CPU seconds) taken by FaCT++ to complete each phase.

In addition to the ontologies, we used artificially generated test data from the DL'98 test suite. Artificial tests are in some sense corner cases for a DL reasoner designed primarily for ontology reasoning, and these tests are mainly intended to investigate the effect of hard problems with very artificial structures on the behaviour of our heuristics. For this purpose we selected from the test suite several of the tests that proved to be hard for FaCT++.

[2]This ontology therefore has a much weaker claim to being "real-life".

Each of these tests consists of a set of 21 satisfiability testing problems of similar structure, but (supposedly exponentially) increasing difficulty; the idea of the test is to determine the number of the largest problem that can be solved within a fixed amount of processing time (100 seconds of CPU time in our case). The names of the tests are of the form "`test_p`" or "`test_n`", where "`test`" refers to the kind of problem (e.g., the "`ph`" tests are derived from encodings of pigeon hole sorting problems), and "`p`/`n`" refers to whether the problems in the test set are satisfiable (`n`) or unsatisfiable (`p`). For these tests we have reported the number of the largest problem solved in less than 100 seconds (21 means that all the problems were solved), along with the time (in CPU seconds) taken for the hardest problem that was successfully solved.

For all the tests, FaCT++ v.0.99.2 was used on Pentium 4 2.2 GHz machine with 512Mb of memory, running Linux. Times were averaged over 3 test runs.

3.3.1 Rule-ordering Heuristics

In these tests we tried a range of different rule-ordering strategies. Each "strategy" is shown as a sequence of letters specifying the priorities (highest first) of the different rule types, where "O" refers to the $\sqcup$-rule, "E" to the $\exists$-rule, and "a" to any other rule type. E.g., "aO" describes the strategy where the $\sqcup$-rule has the lowest priority, and all other rules have an equal higher priority.

Ontology tests The results of using different rule-ordering strategies with the various ontologies are shown in Table 3.7. All ontologies were tested with the best disjunction-ordering heuristic, as determined in separate tests (see below).

KB	DOLCE		WineFood		GALEN	
	SAT	SUB	SAT	SUB	SAT	SUB
a	0.74	0.74	0.22	2.44	99.44	1678.11
aO	0.64	0.68	**0.14**	**1.64**	29.80	569, 64
aEO	**0.58**	**0.57**	0.15	1.67	**9.88**	**173.79**
aE	0.60	0.58	0.27	2.87	13.35	205.32
aOE	0.61	0.59	0.27	2.93	13.22	201.40

Table 3.1: Ontology tests with different rule-orderings

The first thing to note is that rule-orderings have relatively little effect on the DOLCE and WineFood ontologies; in contrast, the performance of the best and worst strategies differs by a factor of almost 10 in the GALEN tests. Even in the GALEN case, however, the difference between the "-O" strategies (i.e., those that assign the lowest priority to the $\sqcup$-rule) and "-E" strategies (i.e., those that assign the lowest priority to the $\exists$-rule) is relatively small. In most cases the best result is given by the "aEO" strategy, i.e., by assigning the lowest priority to the $\sqcup$-rule and the next lowest priority to the $\exists$-rule, and even when "aEO" is not the best strategy, the difference between it and the best strategy is very small. Moreover, the difference between the “aEO” and “aOE” strategies is small in most cases, and never more than a factor of 2.

DL98 tests The results of using different rule-ordering strategies with the DL98 tests are shown in Table 3.2. The first thing to note from these results is that rule-ordering heuristics can have a much more significant effect than in the ontology tests: in some cases the performance of the best and worst strategies differs by a factor of more than 100. In most tests, the “-E” strategies give the best results, with the difference between “-O” and “-E” strategies being much more marked than in the case of the ontology tests. In the case of the `d4_n` test, however, performance is dramatically improved (by a factor of 20) when an “-O” strategy is used.

test	`br_n`		`br_p`		`d4_n`		`ph_n`		`ph_p`	
	last	time	last	time	last	time	last	time	last	time
a	8	16.7	9	20.5	20	94.8	11	99.0	7	15.5
aO	11	38.2	11	38.1	21	0.8	10	10.8	7	32.1
aEO	11	38.8	11	39.0	21	**0.8**	10	10.9	7	32.9
aE	11	**17.1**	12	**18.3**	21	15.7	11	**97.4**	7	**15.2**
aOE	11	19.3	12	21.1	21	16.1	11	99.5	7	15.9

Table 3.2: DL-98 tests with different rule-ordering strategies

3.3.2 Expansion-ordering Heuristics

In these tests we tried a range of different expansion-ordering heuristics. Each heuristic is denoted by two letters, the first of which indicates whether the ordering is based on concept size (“S”), maximum depth (“D”) or frequency of usage (“F”), and the second of which indicates ascending (“a”) or descending (“d”) order. In each group of tests we used the best rule-ordering heuristic as determined by the tests in Section 3.3.1.

Ontology tests For the ontology tests, we tried different orderings for the SAT and SUB phases of classification. The results are presented in Tables 3.3, 3.4 and 3.5; the first figure in each column is the time taken by the SAT phase using the given ordering, and the remaining figures are the subsequent times taken using different SUB phase orderings.

For DOLCE (Table 3.3), the difference between the best and worst orderings was a factor of about 4, and many possible orderings were near optimal. For WineFood (Table 3.4), the difference between the best and worst orderings was a factor of about 2, and using Sd for SAT tests and Dd for SUB tests gave the best result, although several other orderings gave similar results. For GALEN (Table 3.5), the difference between the best and worst orderings was so large that we were only the orderings given allowed tests to be completed in a reasonable time. The best result was given by using Da for both phases.

DL98 tests Table 3.6 presents the results for the DL98 tests. Each column shows the times taken using different expansion orderings to solve the hardest problem that was solvable within the stipulated time limit using any ordering.

SAT	Sa	Da	Fa	Sd	Dd	Fd
SUB	1.29	1.28	1.24	0.61	0.6	0.6
Sa	2.53	2.52	2.52	2.46	2.45	2.41
Da	2.53	2.53	2.53	2.44	2.44	2.41
Fa	0.91	0.91	0.89	0.97	0.98	0.88
Sd	0.61	0.60	0.60	**0.59**	**0.59**	**0.59**
Dd	0.60	0.60	0.60	**0.60**	**0.59**	**0.60**
Fd	1.33	1.34	1.33	1.30	1.34	1.33

Table 3.3: DOLCE test with different expansion-orderings

SAT	Sa	Da	Fa	Sd	Dd	Fd
SUB	0.26	0.29	0.19	0.13	0.13	0.20
Sa	3.15	3.57	3.27	3.21	3.21	3.68
Da	3.54	3.57	3.44	3.20	3.40	3.47
Fa	3.67	3.57	2.32	2.12	2.41	2.35
Sd	1.77	1.80	1.71	1.80	1.80	1.83
Dd	1.69	1.77	1.87	**1.66**	1.78	1.78
Fd	2.30	2.26	2.75	3.14	3.54	2.76

Table 3.4: WineFood test with different expansion-orderings

SAT	Sa	Da
SUB	18.76	9.88
Sa	276.90	276.16
Da	185.79	**172.89**
Fd	1049.74	943.06

Table 3.5: GALEN test with different expansion-orderings

In almost every test, the difference between the best and worst strategies is large: a factor of more than 300 in the d4_n test. Moreover, strategies that are good in one test can be very bad in another (the Sd and Dd strategies are the best ones in the branch tests (br_n and br_p), but (by far) the worst in the d4_n test), and this is not strongly dependent on the satisfiability result (in the br tests, all strategies perform similarly in both satisfiable and unsatisfiable cases). The Fd strategy is, however, either optimal or near optimal in all cases.

3.3.3 Analysis

The different rule-ordering heuristics we tried had relatively little effect on the performance of the reasoner when classifying the DOLCE and WineFood ontologies. With the GALEN ontology, any strategy that gave a lower priority to the $\exists$- and $\sqcup$-rules worked reasonably well, and the aEO strategy was optimal or near-optimal in all cases. The crucial factor with GALEN is giving low priority to the $\exists$-rule. This is due to the fact that GALEN is large, contains many GCIs and also contains existential cycles in concept in-

order	br_n	br_p	d4_n	ph_n	ph_p
	test 11	test 12	test 21	test 10	test 7
Sa	22.6	24.8	0.9	8.1	29.5
Da	22.6	24.8	0.9	$>$300	24.5
Fa	$>$300	$>$300	32.0	22.9	20.2
Sd	**17.0**	**18.3**	$>$300	38.7	24.7
Dd	**17.1**	**18.3**	$>$300	19.7	19.3
Fd	22.2	25.1	**0.8**	**6.2**	**15.3**

Table 3.6: DL98 tests with different Or strategies

clusion axioms (e.g., $C \sqsubseteq \exists R.D$ and $D \sqsubseteq \exists R^{-}.C$); as a result, the graph can grow very large, and this increases both the size of the search space (because GCI related non-determinism may apply on a per-node basis) and the cost of saving and restoring the state during backtracking search. Giving a low priority to the $\exists$-rule minimises the size of the graph and hence can reduce both the size of the search space and the cost of saving and restoring. This effect is less noticeable with the other ontologies because their smaller size and/or lower number of GCIs greatly reduces the maximum size of graphs and/or search space. In view of these results, FaCT++'s default rule-ordering strategy has been set to aEO.[3]

The picture is quite different in the case of the DL'98 tests. Here, different strategies can make a large difference, and no one strategy is universally near optimal. This is to be expected, given that some of the tests include very little non-determinism, but are designed to force the construction of very large models (and hence graphs), while others are highly non-deterministic, but have only very small models. Given that these extreme cases are not representative of typical real-life ontologies, the test results may not be directly relevant to a system designed to deal with such ontologies. It is interesting, however, to see how *badly* the heuristics can behave in such cases: in fact the standard aEO strategy is near optimal in two of the tests, and is never worse than the optimal strategy by a factor of more than 2.

The expansion-ordering heuristics had a much bigger effect on ontology reasoning performance (than the rule-ordering heuristics). In the case of DOLCE and WineFood, almost any strategy that uses Sd or Dd in the SUB phase is near optimal. For GALEN, however, using Da in both phases gives by far the best results. This is again due to the characteristic structure of this ontology, and the fact that preferentially choosing concepts with low modal depth tends to reduce the size of the graph. Unfortunately, no one strategy is universally good (Da/Da is best for GALEN but worst for DOLCE and WineFood); currently, Sd/Dd is the default setting, as the majority of real life ontologies resemble DOLCE and WineFood more than GALEN), but this can of course be changed by the user if it is known that the ontology to be reasoned with will have a GALEN-like structure.

[3]Top-down architectures necessarily give lowest priority to the $\exists$-rule, and generally give low priority to $\sqcup$-rule, which is why they work relatively well with ontologies.

For the DL'98 tests, the picture is again quite confused: the Sd strategy (the default in the SAT phase) is optimal in some tests, but bad in others—disastrously so in the case of the d4_n test. As in the ontology case, the only "solution" offered at present is to allow users to tune these settings according to the problem type or empirical results.

3.4 Discussion and Future Work

We have described the ToDo list architecture used in the FaCT++ system along with a range of heuristics that can be used for rule and expansion ordering. We have also presented an empirical analysis of these heuristics and shown how these have led us to select the default setting currently used by FaCT++.

These default settings reflect the current predominance of relatively small and simply structured ontologies. This may not, however, be a realistic picture of the kinds of ontology that we can expect in the future: many existing ontologies (including, e.g., Wine-Food) pre-date the development of OWL, and have been translated from less expressive formalisms. With more widespread use of OWL, and the increasing availability of sophisticated ontology development tools, it may be reasonable to expect the emergence of larger and more complex ontologies. As we have seen in Section 3.3.1, heuristics can be very effective in helping us to deal efficiently with such ontologies, but choosing a suitable heuristic becomes of critical importance.

In our existing implementation, changing heuristics requires the user to set the appropriate parameters when using the reasoner. This is clearly undesirable at best, and unrealistic for non-expert users. We are, therefore, working on techniques that will allow us to guess the most appropriate heuristics for a given ontology. The idea is to make an initial guess based on an analysis of the syntactic structure of the ontology (it should be quite easy to distinguish GALEN-like ontologies from DOLCE and WineFood-like ontologies simply by examining the statistics that have already been gathered for use in expansion-ordering heuristics), with subsequent adjustments being made based on the behaviour of the algorithm (e.g., the size of graphs being constructed).

Another limitation of the existing implementation is that a single strategy is used for all the tests performed in the classification process. In practice, the characteristics of different tests (e.g., w.r.t. concept size and/or satisfiability) may vary considerable, and it may make sense to dynamically switch heuristics depending on the kind of test being performed. This again depends on having an effective (and cheap) method for analysing the likely characteristics of a given test, and syntactic and behavioural analyses will also be investigated in this context.

KB	DOLCE		WineFood		GALEN	
	SAT	SUB	SAT	SUB	SAT	SUB
a	$\frac{0.74}{388,991}$	$\frac{0.74}{393,067}$	$\frac{0.22}{151,862}$	$\frac{2.44}{1,647,946}$	$\frac{99.44}{36,650,895}$	$\frac{1678.11}{601,211,001}$
aO	$\frac{0.64}{374,937}$	$\frac{0.68}{392,763}$	$\frac{\mathbf{0.14}}{\mathbf{77,563}}$	$\frac{\mathbf{1.64}}{\mathbf{970,377}}$	$\frac{29.80}{10,872,396}$	$\frac{569,64}{199,269,579}$
aEO	$\frac{\mathbf{0.58}}{350788}$	$\frac{\mathbf{0.57}}{339105}$	$\frac{0.15}{78132}$	$\frac{1.67}{977314}$	$\frac{\mathbf{9.88}}{\mathbf{3,809,184}}$	$\frac{\mathbf{173.79}}{\mathbf{64,980,980}}$
aE	$\frac{0.60}{\mathbf{349474}}$	$\frac{0.58}{\mathbf{325726}}$	$\frac{0.27}{166282}$	$\frac{2.87}{1867639}$	$\frac{13.35}{5,828,901}$	$\frac{205.32}{87,152,612}$
aOE	$\frac{0.61}{350165}$	$\frac{0.59}{330233}$	$\frac{0.27}{161152}$	$\frac{2.93}{1900658}$	$\frac{13.22}{5,719,005}$	$\frac{201.40}{85,216,183}$

Table 3.7: Ontology tests with different rule-orderings

Chapter 4

FaCT-DG: Extending FaCT with Datatype Groups

4.1 Introduction

The OWL Web Ontology Language [BvHH$^+$04] is a W3C recommendation for expressing ontologies in the Semantic Web. Datatype support is one of the most useful features OWL is expected to provide, and has brought extensive discussions in the RDF-Logic mailing list [RDF01] and Semantic Web Best Practices mailing list [Sem04]. Recently, Pan and Horrocks [PH05] propose a small extension of OWL DL, called OWL-Eu, to add customised datatypes into OWL DL. Pan [Pan04] proposes a further extension of OWL DL to support *both* customised datatypes *and* datatype predicates. All these languages are indeed based on Description Logics (DLs) integrated with datatype groups. In this paper, we investigate implementations of DL systems that support these DLs.

The main contributions of the paper are as follows. Firstly, we propose a system architecture of DL reasoners for DLs integrated with datatype groups and investigate the flexibility of the architecture (Section 4.3). Secondly, we describe out implement of an extension (called FaCT-DG) of the well known FaCT DL reasoner, using the tableaux algorithm presented in [Pan04] (Section 4.5). The FaCT-DG system supports the $\mathcal{SHIQ}(\mathcal{G})$ DL (Section 4.2.1), and it allows users to use datatype expressions to represent customised datatypes and datatype predicates. Furthermore, we describe how to extend the general API interface DIG/1.1 (Section 4.2.2) of DL systems to DIG/OWL-E, which is compatible with both OWL DL and OWL-E. Last but not least, we present a case study of the FaCT-DG system (Section 4.6).

4.2 Preliminary

4.2.1 The $\mathcal{SHIQ}(\mathcal{G})$ DL

The $\mathcal{SHIQ}(\mathcal{G})$ DL is a sub-language of OWL-E and was first introduced in [Pan04]. The $\mathcal{SHIQ}(\mathcal{G})$ DL extends the $\mathcal{SHIQ}$ DL with an arbitrary datatype group [Pan04]. Intuitively speaking, a datatype group is a group of built-in predicate URIrefs 'wrapped' around a set of primitive datatype URIrefs; it provides a unified formalism for datatypes and datatype predicates. In a datatype group, datatype expressions can be used to represent customised datatypes and datatype predicates. $\mathcal{SHIQ}(\mathcal{G})$ allows the use Datatype expressions can be used in its datatype group-based concept descriptions.

$\mathcal{SHIQ}(\mathcal{G})$ consists of an alphabet of distinct concept names ($\mathbf{C}$), role names ($\mathbf{R}$) and individual (object) names ($\mathbf{I}$); together with a set of constructors to construct concept and role descriptions (also called $\mathcal{SHIQ}(\mathcal{G})$*-concepts* and $\mathcal{SHIQ}(\mathcal{G})$*-roles*, respectively). Now we briefly introduce $\mathcal{SHIQ}(\mathcal{G})$-roles, $\mathcal{SHIQ}(\mathcal{G})$-concepts and $\mathcal{SHIQ}(\mathcal{G})$-RBox as follows. Let $RN \in \mathbf{R}_A$, $R \in \mathbf{Rdsc}_A(\mathcal{SHIQ}(\mathcal{G}))$, $TN \in \mathbf{R_D}$ and $T \in \mathbf{Rdsc_D}(\mathcal{SHIQ}(\mathcal{G}))$. Valid $\mathcal{SHIQ}(\mathcal{G})$ abstract roles are defined by the abstract syntax: $R ::= RN \mid R^-$, where for some $x, y \in \Delta^\mathcal{I}$, $\langle x, y\rangle \in R^\mathcal{I}$ iff $\langle y, x\rangle \in R^{-\mathcal{I}}$. The inverse relation of roles is symmetric, and to avoid considering the roles such as R^{--}, we define a function Inv which returns the inverse of a role, more precisely

$$\mathsf{Inv}(R) := \begin{cases} RN^- & \text{if } R = RN, \\ RN & \text{if } R = RN^-. \end{cases}$$

Valid $\mathcal{SHIQ}(\mathcal{G})$ concrete roles are defined by the abstract syntax: $T ::= TN$. Let $\mathsf{CN} \in \mathbf{C}$, $R \in \mathbf{Rdsc}_A(\mathcal{SHIQ}(\mathcal{G}))$, $T_1, \ldots, T_n \in \mathbf{Rdsc_D}(\mathcal{SHIQ}(\mathcal{G}))$ and $T_i \not\sqsubseteq T_j, T_j \not\sqsubseteq T_i$ for all $1 \leq i < j \leq n$, $C, D \in \mathbf{Cdsc}(\mathcal{SHIQ}(\mathcal{G}))$, $E \in \mathbf{Dexp}(\mathcal{G})$, $n, m \in \mathbb{N}$, $n \geq 1$. Valid $\mathcal{SHIQ}(\mathcal{G})$-concepts are defined by the abstract syntax:

$$\begin{array}{rcl} C & ::= & \top \mid \bot \mid \mathsf{CN} \mid \neg C \mid C \sqcap D \mid C \sqcup D \mid \exists R.C \mid \forall R.C \mid \geqslant mR.C \mid \leqslant mR.C \\ & & \exists T_1, \ldots, T_n.E \mid \forall T_1, \ldots, T_n.E \mid \geqslant mT_1, \ldots, T_n.E \mid \leqslant mT_1, \ldots, T_n.E. \end{array}$$

The semantics of datatype group-based $\mathcal{SHIQ}(\mathcal{G})$-concepts is given in Table 4.1; the syntax and semantic of $\mathcal{SHIQ}(\mathcal{G})$ datatype expressions is presented in Table 4.2. The reader is referred to Section 4.6 for some examples of $\mathcal{SHIQ}(\mathcal{G})$-concepts.

Let $R_1, R_2 \in \mathbf{Rdsc}_A(\mathcal{SHIQ}(\mathcal{G})), T_1, T_2 \in \mathbf{Rdsc_D}(\mathcal{SHIQ}(\mathcal{G})), SN \in \mathbf{R}$, a $\mathcal{SHIQ}(\mathcal{G})$ RBox $\mathcal{R}$ is a finite, possibly empty, set of role axioms:

- functional role axioms $\mathsf{Func}(SN)$;
- transitive role axioms $\mathrm{Trans}(R_1)$;[1]

[1]Note that a concrete role T can not be a transitive role.

Constructor	DL Syntax	Semantics
top	$\top$	$\Delta^{\mathcal{I}}$
bottom	$\bot$	$\emptyset$
concept name	CN	$\mathsf{CN}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
general negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
disjunction	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
nominals	$\{\mathsf{o}\}$	$\{\mathsf{o}\}^{\mathcal{I}} = \{\mathsf{o}^{\mathcal{I}}\}$
exist restriction	$\exists R.C$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y.\langle x,y\rangle \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
value restriction	$\forall R.C$	$\{x \in \Delta^{\mathcal{I}} \mid \forall y.\langle x,y\rangle \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$
atleast restriction	$\geqslant mR.C$	$\{x \in \Delta^{\mathcal{I}} \mid \sharp\{y \mid \langle x,y\rangle \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} \geq m\}$
atmost restriction	$\leqslant mR.C$	$\{x \in \Delta^{\mathcal{I}} \mid \sharp\{y \mid \langle x,y\rangle \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} \leq m\}$
expressive predicate exists restriction	$\exists T_1,\ldots,T_n.E$	$\{x \in \Delta^{\mathcal{I}} \mid \exists t_1,\ldots,t_n.\langle x,t_i\rangle \in T^{\mathcal{I}}$ (for all $1 \leq i \leq m) \wedge \langle t_1,\ldots,t_n\rangle \in E^{\mathbf{D}}\}$
expressive predicate value restriction)	$\forall T_1,\ldots,T_n.E$	$\{x \in \Delta^{\mathcal{I}} \mid \forall t_1,\ldots,t_n.\langle x,t_i\rangle \in T^{\mathcal{I}}$ (for all $1 \leq i \leq m) \rightarrow \langle t_1,\ldots,t_n\rangle \in E^{\mathbf{D}}\}$
expressive predicate atleast restriction	$\geqslant mT_1,\ldots,T_n.E$	$\{x \in \Delta^{\mathcal{I}} \mid \sharp\{\langle t_1,\ldots,t_n\rangle \mid \langle x,t_i\rangle \in T^{\mathcal{I}}$ (for all $1 \leq i \leq m) \wedge \langle t_1,\ldots,t_n\rangle \in E^{\mathbf{D}}\} \geq m\}$
expressive predicate atmost restriction	$\leqslant mT_1,\ldots,T_n.E$	$\{x \in \Delta^{\mathcal{I}} \mid \sharp\{\langle t_1,\ldots,t_n\rangle \mid \langle x,t_i\rangle \in T^{\mathcal{I}}$ (for all $1 \leq i \leq m) \wedge \langle t_1,\ldots,t_n\rangle \in E^{\mathbf{D}}\} \leq m\}$

Table 4.1: $\mathcal{SHIQ}(\mathcal{G})$ concept descriptions

Abstract Syntax	DL Syntax	Semantics
rdfs:Literal	$\top_{\mathrm{D}}$	$\Delta_{\mathbf{D}}$
owlx:DatatypeBottom	$\bot_{\mathrm{D}}$	$\emptyset$
u a predicate URIref	u	$u^{\mathbf{D}}$

`not`(u)	$\overline{u}$	if $u \in \mathbf{D}_{\mathcal{G}},\ \Delta_{\mathbf{D}} \setminus u^{\mathbf{D}}$
		 if $u \in \Phi_{\mathcal{G}} \setminus \mathbf{D}_{\mathcal{G}},\ (\mathsf{dom}(u))^{\mathbf{D}} \setminus u^{\mathbf{D}}$
		 if $u \notin \Phi_{\mathcal{G}},\ \bigcup_{n\geq 1}(\Delta_{\mathbf{D}})^n \setminus u^{\mathbf{D}}$

`oneOf`$(``s_1"\hat{}\hat{}d_1 \ldots ``s_n"\hat{}\hat{}d_n)$	$\{``s_1"\hat{}\hat{}d_1,\ldots,``s_n"\hat{}\hat{}d_n\}$	$\{(``s_1"\hat{}\hat{}d_1)^{\mathbf{D}}\} \cup \cdots \cup \{(``s_n"\hat{}\hat{}d_n)^{\mathbf{D}}\}$
`domain`$(v_1,\ldots,v_n)$	$[v_1,\ldots,v_n]$	$v_1^{\mathbf{D}} \times \cdots \times v_n^{\mathbf{D}}$
`and`(P,Q)	$P \wedge Q$	$P^{\mathbf{D}} \cap Q^{\mathbf{D}}$
`or`(P,Q)	$P \vee Q$	$P^{\mathbf{D}} \cup Q^{\mathbf{D}}$

Table 4.2: $\mathcal{SHIQ}(\mathcal{G})$ datatype expressions

- abstract role inclusion axioms $R_1 \sqsubseteq R_2$;
- concrete role inclusion axioms $T_1 \sqsubseteq T_2$.

We extend $\mathcal{R}$ to the *role hierarchy* of $\mathcal{SHIQ}(\mathcal{G})$ as follows:

$$\mathcal{R}^+ := (\mathcal{R} \cup \{\mathsf{Inv}(R) \sqsubseteq \mathsf{Inv}(S) \mid R \sqsubseteq S \in \mathcal{R}\},\ \underline{\overset{*}{\sqsubseteq}})$$

where $\underline{\overset{*}{\sqsubseteq}}$ is the transitive-reflexive closure of $\sqsubseteq$ over $\mathcal{R} \cup \{\mathsf{Inv}(R) \sqsubseteq \mathsf{Inv}(S) \mid R \sqsubseteq S \in \mathcal{R}\}$. The tableaux algorithm for the $\mathcal{SHIQ}(\mathcal{G})$ DL is presented in [Pan04].

4.2.2 DIG/1.1 Interface

DIG/1.1 is a simple API for a general DL system developed by the DL Implementation Group (DIG). There is a commitment from the implementors of the leading DL reasoners (FaCT [Hor98b], Racer [HM01d] and Cerebra[2]) to provide implementations conforming to the DIG/1.1 specification. There are four kinds of requests from clients and a server:

1. Clients can find out which reasoner is actually behind the interface by sending an `IDENTIFIER` request. The server then returns the name and the version of the reasoner, together with the list of constructors, `TELL` and `ASK` operations supported by the reasoner.

2. Clients can request to add or release knowledge base URIrefs as a local identifiers[3] for `TELL` and `ASK` requests.

3. Clients can send `TELL` requests using the `TELL` language of DIG/1.1 The `TELL` requests are monotonic, i.e., DIG/1.1 does not support removing the `TELL` requests that have been sent – the only work-around is to release the knowledge base and start again. A `TELL` request must be made in the context of a particular knowledge base (by using the knowledge base URIref). The server will response using the basic responses.

4. Clients can send `ASK` requests using the `ASK` language of DIG/1.1. An `ASK` request can contain multiple queries (with different IDs) and must be made in the context of a particular knowledge base (by using the knowledge base URIref). The server will response using the `RESPONSE` language of DIG/1.1.

The main drawback of DIG/1.1 is that it does not completely support OWL DL and OWL-E.

4.3 Architecture

In this section, we propose a system architecture (see Figure 4.1) of DL systems for DLs integrated with datatype groups. Here we use DIG/OWL-E (see Section 4.4) instead of DIG/1.1 as the general DL reasoner interface in the new architecture.

There are two kinds of datatype reasoners in the framework. The first kind of datatype reasoners are called datatype managers, which provide decision procedures to reduce the satisfiability problem of datatype expression conjunctions to the satisfiability problems of various datatype predicate conjunctions. Intuitively, a datatype manager transforms an input datatype expression into a disjunction of predicate conjunctions, and then divides each

[2]`http://www.networkinference.com`

[3]A local identifier is reasoner-dependent; cf. [Bec03].

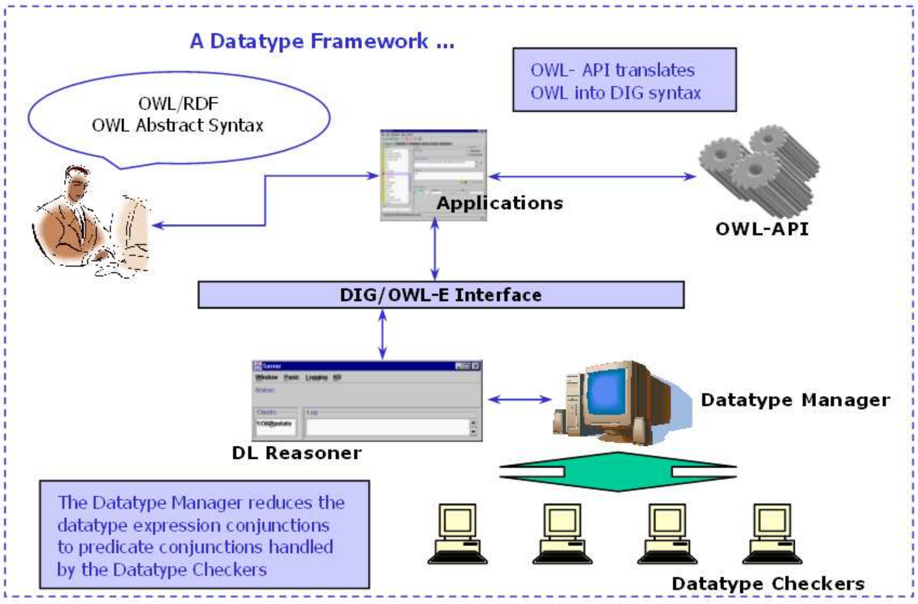

A Datatype Framework ...
OWL/RDF
OWL Abstract Syntax
OWL- API translates OWL into DIG syntax
Applications
OWL-API
DIG/OWL-E Interface
Datatype Manager
DL Reasoner
The Datatype Manager reduces the datatype expression conjunctions to predicate conjunctions handled by the Datatype Checkers
Datatype Checkers

Figure 4.1: Framework Architecture

disjunct into several sub-conjunctions. Each of these sub-conjunctions contains predicates of certain base datatype. If there exist variables being used across these sub-conjunctions, then the corresponding disjunct is *unsatisfiable*, as the value spaces of base datatypes are disjoint with each other; otherwise, the datatype manager can send these sub-conjunctions to proper datatype checkers to decide their satisfiabilities. If all the sub-conjunctions of a disjunct is *satisfiable*, then the input datatype expression is *satisfiable*.

The second kind of datatype reasoner are called datatype checkers. A datatype checker decides the satisfiability problem of datatype predicate conjunctions, where the datatype predicates are defined over a base datatype in a datatype group. More technical details about the datatype managers and datatype checkers implemented in FaCT-DG will be presented in Section 4.5.3.

As shown in Figure 4.1, when a client sends an identification request to the DIG/OWL-E server, the server returns the names and versions of the DL reasoner, the datatype manager and the datatype checkers. In addition, the server returns the description language that the DL reasoner supports, the datatype expressions that the datatype manager supports, and the base datatype URIrefs and supported predicate URIrefs that the datatype checkers support.

Then the client can upload the knowledge base using the TELL language and query using the ASK language. To answer the query, the DL reasoner runs as usual, and asks the datatype manager to decide the satisfiability problem of datatype expression conjunctions whenever necessary. The datatype manager reduces the datatype expression conjunctions to predicate conjunctions, and then pass them to the proper datatype checkers to check

Typed Literals	`<typedValue lxform = "L" datatype = "DN"/>`

Datatype Expressions	`<dttop/>`
	`<dtbottom/>`
	`<predicate name = "PN"/>`
	`<dtnot name = "PN"/>`
	`<vset>` $V_1 \ldots V_n$ `</vset>`
	`<dtand>` $E_1 \ldots E_n$ `</dtand>`
	`<dtor>` $E_1 \ldots E_n$ `</dtor>`
	`<dtdomain>` $U_1 \ldots U_n$ `</dtdomain>`
	`<dtexpression name = "EN"/>`

Concrete Roles	`<dtratom name = "TN"/>`

Datatype Expression-related Concept Descriptions	`<dtsome>` $T_1 \ldots T_n$ E `</dtsome>`
	`<dtall>` $T_1 \ldots T_n$ E `</dtall>`
	`<dtatmost num = "n">` $T_1 \ldots T_n$ E `</dtatmost>`
	`<dtatleast num = "n">` $T_1 \ldots T_n$ E `</dtatleast>`

Table 4.3: New constructors in the DIG/OWL-E description language

their satisfiability. Finally, the DL reasoner uses the RESPONSE language to return the answer to the client.

In case we need to support a new form of datatype expression, we should update our datatype manager; while in case we need to provide some new datatype predicates that are defined over a new base datatype, then we should add a new datatype checker.[4] In either cases, we do not have to update the DL reasoner.

4.4 DIG/OWL-E

To overcome the limitations of DIG/1.1 presented at the end of the last section, we propose DIG/OWL-E, which supports OWL-E (viz. $\mathcal{SHOIQ}(\mathcal{G})$) and extends the DIG/1.1 description language with the following constructors (see Table 4.3):[5]

1. **Typed Literals**: A `<typedValue>` element represents a typed literal; e.g.,

   `<typedValue lxform = "s" datatype = "u"/>`

   represents the typed literal "s"ˆˆu.

2. **Datatype Expressions**:

[4]Value spaces of the base datatypes should be disjoint with each other.

[5]Notations of Table 4.3: $\mathtt{V_i}$ are typed literals, $\mathtt{E_i}$ are datatype expressions or datatype expression names, $\mathtt{U_i}$ are unary datatype predicates or their relativised negations, and $\mathtt{T_i}$ are concrete roles.

(a) `<dttop/>` and `<dtbottom/>` correspond to $\mathrm{rdfs{:}Literal}$ and $\mathrm{owlx{:}Datatype}$-Bottom in OWL-E, respectively.

(b) A `<predicate>` element introduces a predicate URI reference, and a `<dtnot>` element represents a negated predicate URI reference.

(c) A `<vset>` element represents an enumerated datatype, i.e., a datatype defined by enumerating all its member typed literals.

(d) A `<dtnot>` element represents the relativised negation of a unary supported predicate URIref.

(e) The `<dtand>`, `<dtor>` and `<dtdomain>` elements corresponds to the `and`, `or` and `domain` constructors defined in Definition **??**, respectively.

3. **Concrete Roles**: A `<dtratom>` element represents a concrete role. Note that attributes (`<attribute>` elements) are simply functional concrete roles.

4. **Datatype Expression-related Concept Descriptions**: `<dtsome>`, `<dtall>`, `<dtatleast>` and `<dtatmost>` elements corresponds to expressive predicate exists restriction ($\exists T_1, \ldots, T_n.E$), expressive predicate value restriction ($\forall T_1, \ldots, T_n.E$), expressive predicate qualified atleast restriction ($\leqslant mT_1, \ldots, T_n.E$) and expressive predicate qualified atmost restriction ($\leqslant mT_1, \ldots, T_n.E$), respectively; note that E can either be a datatype expression or a datatype expression name.

Example 1 *A Concept Description in DIG/OWL-E*
The AdultElephant *concept can be defined in DIG/OWL-E as follows:*

```
<tells ...>
  <equalc>
    <catom name = "AdultElephant"/>
    <and>
      <catom name = "Elephant"/>
      <dtsome>
        <attribute name = "age"/>
        <predicate id = "owlx:integerGreaterThanx=20"/>
      </dtsome>
    </and>
  </equalc>
  <defconcept name = "Elephant"/>
  <defattribute name = "age"/>
</tells>
```

where $\mathrm{owlx{:}integerGreaterThanx{=}20}$ *is the URIref for the integer predicate* $>^{int}_{[20]}$. *Here we use the* `<dtsome>` *construct, instead of the* `<intmin>` *construct; the main benefit is that we can now use arbitrary predicate URIrefs, or even datatype expressions.* ♢

DIG/OWL-E extends the DIG/1.1 `TELL` language by introducing the following new axioms (see Table 4.4 on page 31):

1. **Datatype Expression Axiom**: A `<defdtexpression>` element represents a datatype expression axiom, which introduces a name "EN" for a datatype expression E. Therefore, the datatype expression name can be used in datatype expression-related concept descriptions and the concrete role range axioms.

Datatype Expression Axiom	<defdtexpression name = "EN"> E </defdtexpression>
Concrete Role Axioms	<defcrole name = "RN"/> <impliescr> T_1 T_2 </impliescr> <equalcr> T_1 T_2 </equalcr> <crdomain> T C </crdomain> <crrange> T E </crrange> <crfunctional> T </crfunctional>
Individual Axioms	<sameindividual> I_1 ... I_n </sameindividual> <diffindividual> I_1 ... I_n </diffindividual>

Table 4.4: New axioms in the DIG/OWL-E TELL language

2. **Concrete Role Axioms**: Many of the concrete role axioms are very similar to abstract role axioms. We could have modified the existing abstract role axioms to accommodate the concrete role axioms. We propose otherwise in order to maintain backward compatibility. Another advantage is that we can easily disallow asserting that an abstract role is a sub-role of (or equivalent to) a concrete one, or the other way around. Note that the concrete role range axiom (represented by a <crrange> element) is quite different from the abstract role range axiom (represented by a <range> element) in that the range in the former one is a datatype expression or a name of a datatype expression, instead of a concept.

3. **Same and Different Individual Axioms**: A <sameindividual> element asserts that two individual names are interpreted as the same individual, while a <diffindividual> element asserts that two individual names are interpreted as different individuals. Note that these two individual axioms are convenient, but not necessary, for people to use OWL and OWL-E.

Example 2 *A Datatype Expression Axiom in DIG/OWL-E*
The sumLessThan15 customised datatype predicate can be defined by the following DIG/OWL-E datatype expression axiom:

```
<tells>
  <defdtexpression name = "sumNoGreaterThan15">
    <dtand>
      <predicate id = "owlx:integerAddition"/>
      <dtdomain>
        <dtnot id = "owlx:integerGreaterThanx=15"/>
        <predicate id = "owlx:integerGreaterThanx=0"/>
        <predicate id = "owlx:integerGreaterThanx=0"/>
        <predicate id = "owlx:integerGreaterThanx=0"/>
      </dtdomain>
    </dtand>
  </defdtexpression>
</tells>
```

where owlx:integerAddition *is the URIref of the integer predicate* $+^{int}$*. The datatype expression* sumNoGreaterThan15 *is a conjunction, where the first conjunct is the predicate*

Individual Query	`<relatedIndividualValues>` T `</relatedIndividualValues>`

Table 4.5: New queries in the DIG/OWL-E ASK language

Individual Value Pair Sets	`<individualValuePairSet>` `<indvalPair>` I_1 V_1 `</indvalPair>` `<indvalPair>` I_1 V_2 `</indvalPair>` `</individualValuePairSet>`
Typed Value Sets	`<typedValueSet>` V_1 … V_n `</typedValueSet>`

Table 4.6: New responses in the DIG/OWL-E RESPONSE language

URIref owlx:integerAddition*, and the second conjunct is a domain constraint, which sets the domains of all its four arguments: the first one (corresponding to the sum of addition) should be integers that are no greater than 15, and the rest are positive integers. Based on* sumNoGreaterThan15*, we can define the* SmallItem *concept as follows:*

```
<tells>
  <defconceptname = "Item"/>
  <defconceptname = "SmallItem"/>
  <equalc>
  <catom name = "SmallItem"/>
    <and>
    <catom name = "Item"/>
      <dtsome>
        <dtratom name = "hlwSumInCM"/>
        <dtratom name = "heightInCM"/>
        <dtratom name = "lengthInCM"/>
        <dtratom name = "widthInCM"/>
        <dtexpression name = "sumNoGreaterThan15"/>
      </dtsome>
    </and>
  </equalc>
</tells>                                                      ◊
```

DIG/OWL-E extends the DIG/1.1 ASK language by introducing a new form of individual query. With a `<relatedIndividualValues>` element, clients can request the instances (pairs of individual and typed values) of a concrete role. Accordingly, DIG/OWL-E extends the DIG/1.1 RESPONSE language (see Table 4.6) with individual value pair sets (`<individualValuePairSet>` elements) and typed value sets. Furthermore, a response to an identifier request in DIG/OWL-E should include the list of base datatype URIrefs, supported predicate URIrefs and the supported datatype expression constructors.

4.5 FaCT-DG

The goal of this section is to describe a datatype extension of the FaCT reasoner [Hor99].[6] FaCT was the first sound and complete DL system to demonstrate the usefulness of expressive Description Logics for developing practical applications [MH03].

Our implementation is designed for two purposes. Firstly, it is meant to be a lightweight test bed for the $\mathcal{G}$-augmented tableaux algorithm of the $\mathcal{SHIQ}(\mathcal{G})$ DL presented [Pan04]. Secondly, it aims to show the flexibility of the framework architecture presented in Section 4.3. As a concept-proof prototype, it provides no implementation of the DIG/OWL-E interface.

4.5.1 System Overview

As per the framework presented in Section 4.3, our prototype is actually a hybrid reasoner that has three highly independent components: (i) a DL reasoner that is built based on the FaCT reasoner and supports the $\mathcal{SHIQ}(\mathcal{G})$ DL (see Section 4.2.1), (ii) a datatype manager which decides the satisfiability of datatype expression conjunctions, and (iii) two simple datatype checkers for integers and strings. The DL reasoner asks the datatype manager to check the satisfiability of datatype queries generated by the tableau algorithm of $\mathcal{SHIQ}(\mathcal{G})$. The datatype manager asks the two datatype checkers to check the satisfiability of predicate conjunctions. Although all the three components of our system is implemented in Lisp, in principle they do *not* have to be implemented in same the programming language.

4.5.2 Extended DL Reasoner

The extended FaCT DL reasoner supports datatype group-based concept descriptions. It implements the tableau algorithm for $\mathcal{SHIQ}(\mathcal{G})$ [Pan04]. We usually apply these rules (in the following order: $\exists_P$-rule, $\geqslant_P$-rule, $\forall_P$-rule, $choose_p$-rule and $\leqslant_P$-rule) on an abstract node just before we further check its abstract successors and query the datatype manager to check resulting datatype constraints. It may need to query the datatype manager again if new datatype constraints are added later, e.g., from some of its abstract successors via some inverse roles.

The DL reasoner is independent of the forms of datatype expressions that the datatype manager supports. It does not have to understand the syntax of datatype expressions, and it simply leaves them untouched and passes them to the datatype manager. Therefore, the DL reasoner does not have to be modified even if the datatype manager is upgraded to support some forms of new datatype expressions.

[6]'FaCT' for 'Fast Classification of Terminologies'.

We extend the FaCT syntax by introducing datatype expressions (see Table 4.7) and datatype expression-related concept descriptions (see Table 4.8), where we use a positive integer n to indicate the number of concrete roles used in the concept descriptions.

Extended FaCT Syntax	OWL-E Abstract Syntax
TOPD	$\mathrm{rdfs{:}Literal}$
BOTTOMD	$\mathrm{owlx{:}DatatypeBottom}$
(and $E_1 \ \ldots \ E_n$)	and($E_1, \ldots, E_n$)
(or $E_1 \ \ldots \ E_n$)	or($E_1, \ldots, E_n$)
(neg p)	neg(p)
(domain $d_1 \ \ldots \ d_n$)	domain($d_1, \ldots, d_n$)

Table 4.7: FaCT datatype expressions

Extended FaCT Syntax	DL Standard Syntax
TOP	$\top$
BOTTOM	$\bot$
(and $\mathsf{C}_1, \ldots, \mathsf{C}_n$)	$\mathsf{C}_1 \sqcap \ldots \sqcap \mathsf{C}_n$
(or $\mathsf{C}_1, \ldots, \mathsf{C}_n$)	$\mathsf{C}_1 \sqcup \ldots \sqcup \mathsf{C}_n$
(not C)	$\neg\mathsf{C}$
(some R C)	$\exists R.\mathsf{C}$
(all R C)	$\forall R.\mathsf{C}$
(atleast m R C)	$\geqslant mR.\mathsf{C}$
(atmost m R C)	$\leqslant mR.\mathsf{C}$
(dt-some n $T_1 \ \ldots \ T_n$ E)	$\exists T_1, \ldots, T_n.E$
(dt-all n $T_1 \ \ldots \ T_n$ E)	$\forall T_1, \ldots, T_n.E$
(dt-atleast n m $T_1 \ \ldots \ T_n$ E)	$\geqslant mT_1, \ldots, T_n.E$
(dt-atmost n m $T_1 \ \ldots \ T_n$ E)	$\leqslant mT_1, \ldots, T_n.E$

Table 4.8: FaCT concepts

The syntax of TBox and RBox axioms remains the same as FaCT. Users can now use datatype expression-related concept descriptions in TBox axioms. Note that, as in the OWL DL ontology language, the set of abstract role names and the set of concrete role names should be disjoint,[7] otherwise the system will report an error. Users can define concrete role inclusion axioms and functional axioms. Note that if users use an abstract role and a concrete role in a role inclusion axiom, the system will report an error. The syntax of TBox queries remains the same as FaCT too; i.e., this datatype extension of FaCT provides concept satisfiability, concept subsumption and classification checking.

[7]By definition they are disjoint because abstract domains are disjoint with datatype domains.

4.5.3 Datatype Reasoning Components

The datatype reasoning components of the hybrid reasoner include a datatype manager and two simple datatype checkers (concrete domain reasoners). The datatype manager reduces the satisfiability problem of datatype expression conjunctions to the satisfiability problem of predicate conjunctions that the datatype checkers can handle.

	Datatype Checker ♯1	**Datatype Checker ♯2**
Sat. Checking Func.	sint-dcf-sat	sstr-dcf-sat
Base Datatype URIref	xsd:integer	xsd:string
Inequality Pred. URIref	owlx:integerInequality	owlx:stringInequality
Other Supported Pred. URIrefs	owlx:integerEquality owlx:integerLessThan owlx:integerGreaterThan owlx:integerLessThanOrEqualTo owlx:integergreaterThanOrEqualTo owlx:integerInequalityx=n owlx:integerEqualityx=n owlx:integerLessThanx=n owlx:integerGreaterThanx=n owlx:integerLessThanOrEqualTox=n owlx:integergreaterThanOrEqualTox=n	owlx:stringEquality

Table 4.9: Registration information of our datatype checkers

The datatype manager is independent of the kinds of datatype predicates that the datatype checkers support. Theoretically it can work with an arbitrary set of datatype checkers, as long as the datatype checkers satisfy the following conditions.

1. the domains of the base datatypes that the datatype checkers support are pairwise disjoint;
2. each datatype checker provides the following registration information:
   (a) its satisfiability checking function,
   (b) its base datatype URIref,
   (c) the inequality predicate URIref for its base datatype,
   (d) the set of supported predicate URIrefs for its base datatype;
3. each registered datatype checker supports the syntax of predicate conjunctions that the datatype manager uses in its queries.

There are several remarks here. Firstly, base datatypes are primitive and are disjoint with each other. If a datatype is more general than another datatype supported in the system, it can not be the base datatype (but a derived one). The disjointness of the value spaces of base datatypes is a useful property for the algorithm of the datatype manager. Secondly, although we assume that each datatype checker supports only one base datatype, it is easy to extend it to the case where a datatype checker supports multiple base datatypes. Furthermore, although we require each registered datatype checker supports the syntax of predicate conjunctions that the datatype manager uses, it should not be difficult to translate the syntax between the datatype manager and datatype checkers.

Table 4.9 lists the registration information of the two datatype checkers implemented in our hybrid reasoner.

It is straightforward, but important, to observe that the system is very flexible. For example, to support new datatypes and predicates, we simply need to add datatype checkers that satisfy the above three conditions.

4.6 Case Study: Match Making

We invite the reader to consider a use case of our hybrid reasoner; i.e., how to use our prototype to support matchmaking.

4.6.1 Matchmaking

Let us consider the following scenario: agents advertise services with their capabilities through a registry and query the registry for services with specified capabilities. Matchmaking is a process that takes a query as input and return all advertisements which may potentially satisfy the capabilities specified in the query.

We can use the $\mathcal{SHIQ}(\mathcal{G})$ DL to describe service capabilities for both advertisement and query. More precisely, the capability (either in advertisements or queries) of a service can be represented as an $\mathcal{SHIQ}(\mathcal{G})$ class or class restriction, e.g. the capability that memory size should be either 256Mb or 512Mb, can be represented the datatype expression-related concept $\exists memoryUnitSizeInMb.(=_{256} \vee =_{512})$.

Usually, we are not only interested in finding the exact match, viz., there could be several degrees of matching. Following [LH03], we consider five levels of matching:

1. **Exact** If the capabilities of an advertisement A and a request R are equivalent classes, we call it an exact match, noted as $C_A \equiv C_R$.

2. **PlugIn** If the capability of a request R is a sub-class of that of an advertisement A, we call it a PlugIn match, noted as $C_R \sqsubseteq C_A$.

3. **Subsume** If the capability of a request R is a super-class of that of an advertisement A, we call it a Subsume match, noted as $C_A \sqsubseteq C_R$.

4. **Intersection** If the intersection of the capabilities of an advertisement A and a request R are satisfiable, we call it a Intersection match, noted as $\neg(C_A \sqcap C_R \sqsubseteq \bot)$.

5. **Disjoint** Otherwise, we call it a Disjoint (failed) match , noted as $C_A \sqcap \ C_R \sqsubseteq \bot$.

4.6.2 Working Examples

To gain a further insight into the above five levels of matching, it is often helpful to have some working examples. Suppose, in a scenario of computer selling, that an agent would like to buy a PC with the following capabilities:

- the $processor$ must be Pentium4;
- the $memoryUnitSizeInMb$ must be 128;
- the $priceInPound$ must be less than 500.

This can be represented by the following $\mathcal{SHIQ}(\mathcal{G})$ concept:

$$\begin{aligned} C_{R1} \quad \equiv \quad & \mathsf{PC} \sqcap \exists processor.\mathsf{Pentium4} \sqcap \\ & \exists memoryUnitSizeInMb. =^{int}_{[128]} \sqcap \exists priceInPound. <^{int}_{[500]} \end{aligned}$$

Figure 4.2 presents five example matching advertisements for C_{R1} in five different matching levels. Among them, C_{A1} is the exact match. In realistic situations, however, it is not to easy have an exact match, since advertisements might provide more general or more specific information. For example, C_{A2} states that $priceInPound$ is only less than 700 and that the $memoryUnitSizeInMb$ can be either 128 or 256 (represented by the datatype expression $=^{int}_{[128]} \vee =^{int}_{[256]}$). C_{A3} adds two restrictions on $orderDate$s: firstly, the order date must be in August and September of 2004, which is represented by the datatype expression $(\geq^{int}_{[20040801]} \wedge \leq^{int}_{[20040831]}) \vee (\geq^{int}_{[20040901]} \wedge \leq^{int}_{[20040930]})$; secondly, $orderDate$s should be sooner than (represented by the binary predicate $<^{int}$) $DeliverDate$s, indicating that PCs will be delivered on some date *after* orders are made. As a result, C_{A2} and C_{A3} are PlugIn match and Subsume match of C_{R1}, respectively. C_{A4} says the $priceInPound$ is greater than 400, and the $CPUFreqInGHz$ of their PCs is 2.8;[8] it is an Intersection match. Finally, C_{A5} advertises that their PCs have exactly two memory chips, with the $memoryUnitSizeInMb$ of each chip is 256, and the $HardDiskBrand$ and $USBKeyBrand$ in their PCs are the same (represented by the binary predicate $=^{str}$); hence it is a disjoint (failed) match.

[8]Note that $=^{real}_{[2.8]}$ is not a supported predicate for our prototype: our prototype will not reject it and even provides minimum checking for it; i.e., if $=^{real}_{[2.8]}$ and $\overline{=^{real}_{[2.8]}}$ are both in a predicate conjunction, this conjunction is *unsatisfiable*.

$$
\begin{array}{rcl}
\text{Exact match: } C_{A1} & \equiv & \mathsf{PC} \sqcap \exists processor.\mathsf{Pentium4} \sqcap \\
 & & \geqslant 1 memoryUnitSizeInMb. =^{int}_{[128]} \sqcap \exists priceInPound. <^{int}_{[500]} \\
\text{PulgIn match: } C_{A2} & \equiv & \mathsf{PC} \sqcap \exists processor.\mathsf{Pentium4} \sqcap \\
 & & \geqslant 1 memoryUnitSizeInMb.(=^{int}_{[128]} \vee =^{int}_{[256]}) \\
 & & \sqcap \exists priceInPound. <^{int}_{[700]} \\
\text{Subsume match: } C_{A3} & \equiv & \mathsf{PC} \sqcap \exists processor.\mathsf{Pentium4} \sqcap \\
 & & \geqslant 1 memoryUnitSizeInMb. =^{int}_{[128]} \sqcap \exists priceInPound. <^{int}_{[500]} \\
 & & \sqcap \forall orderDate.((\geq^{int}_{[20040801]} \wedge \leq^{int}_{[20040831]}) \vee \\
 & & (\geq^{int}_{[20040901]} \wedge \leq^{int}_{[20040930]})) \sqcap \forall orderDate, deliverDate. <^{int} \\
\text{Intrsect. match: } C_{A4} & \equiv & \mathsf{PC} \sqcap \exists processor.\mathsf{Pentium4} \sqcap \\
 & & \geqslant 1 memoryUnitSizeInMb. =^{int}_{[128]} \sqcap \exists priceInPound. >^{int}_{[400]} \\
 & & \sqcap \leqslant 1 priceInPound.\top_{\mathrm{D}} \sqcap \ \exists CPUFreqInGHz. =^{real}_{[2.8]} \\
\text{Disjoint match: } C_{A5} & \equiv & \mathsf{PC} \sqcap \exists processor.\mathsf{Pentium4} \sqcap \\
 & & \geqslant 2 memoryUnitSizeInMb. =^{int}_{[256]} \sqcap \exists priceInPound. <^{int}_{[500]} \\
 & & \leqslant 2 memoryUnitSizeInMb. =^{int}_{[256]} \\
 & & \sqcap \forall HardDiskBrand, USBKeyBrand. =^{str}
\end{array}
$$

Figure 4.2: Example matching advertisements

4.7 Conclusion and Outlook

This chapter proposes a flexible framework architecture to provide reasoning services for Description Logics integrated with datatype groups.

Users The framework enables the users to use customised datatypes and datatype predicates. In addition, it provides minimum checking for unsupported datatype predicates.

DIG/OWL-E interface Our general DL API does not need to be updated when new datatype predicates are supported by datatype reasoners.

DL reasoner In the framework, the DL reasoner can implement $\mathcal{G}$-augmented tableaux algorithms for a wide range of $\mathcal{G}$-combined DLs, including expressive ones like $\mathcal{SHIQ}(\mathcal{G})$, $\mathcal{SHOQ}(\mathcal{G})$ and $\mathcal{SHIO}(\mathcal{G})$, and less express but more efficient ones like the datatype group extensions of DL-Lite [CDGL$^+$04] and $\mathcal{ELH}$ [Bra04]. To support new DLs, we only need to upgrade the DL reasoner.

Datatype Checkers To support datatype predicates of some new base datatypes, we only have to add new datatype checkers.

Many of the above flexibilities have been witnessed by the FaCT-DG system, which is an extension of the FaCT DL reasoner with support for customised datatypes and datatype predicates.

As for future work, on the one hand, we will investigate the complexity of the $\mathcal{SHIQ}(\mathcal{G})$ DL and see if the known optimisation techniques [TH00, VH01, HM02] for concrete domains can be applied in $\mathcal{G}$-augmented tableaux algorithms and the datatype manager algorithm of the FaCT-DG system. On the other hand, we would like to investigate the performance of the FaCT-DG system in some ontology applications.

Chapter 5

Implementing an f-$\mathcal{SI}$ Reasoner

Many applications and domains, nowadays, use some form of knowledge representation language in order to improve their capabilities. Such examples of applications are the World Wide Web [BLHL01] or multimedia applications [ABS03, BSC00]. In the architecture of these applications, ontologies play a key role as they are used as a source of shared and precisely defined terms that can be used in metadata. This led to the creation of suitable ontology languages like OWL and DAML+OIL. Both these languages are based on highly expressive description logics to represent knowledge and support a wide range of reasoning services. Although DLs provide considerable expressive power, they feature expressive limitations regarding their ability to represent vague and imprecise knowledge. The role and significance of handling uncertainty has been pointed out many times in literature, and many applications like multimedia [SST$^+$05], decision making [Zim87], and many more have incorporated mathematical frameworks in order to handle various types of uncertainty. One such important theory is fuzzy set theory. Unfortunately, few work has been done in the context of DLs and the semantic web.

In this chapter we present an extension of the description logic $\mathcal{SI}$ with fuzzy set theory. We provide detailed reasoning algorithms which are both sound and complete.

5.1 Syntax and Semantics of f-$\mathcal{SI}$

In fuzzy $\mathcal{SI}$, f-$\mathcal{SI}$ for short [1], the basic fuzzy DL f-$\mathcal{ALC}$ [Str01] is extended with fuzzy transitive and inverse roles. The set of transitive roles $\mathbf{R}_+$ is a subset of the set of roles $\mathbf{R}$. In addition, for any role $R \in \mathbf{R}$, the role R^- is interpreted as the inverse of R. Similarly to [HS99] we introduce two functions. The first one is the function Inv which given a role R it returns its inverse, R^-, and given an inverse role, R^-, it returns the role R. At last, for

[1]In a previous approach to fuzzy DLs the prefix μ is used [Str04] but this letter is reserved by DLs with *fixed point constructors* [BMNPS02]. In some other approaches [TM98, HKS02] the naming $\mathcal{ALC}_F$ is used but this can easily be confused with $\mathcal{ALCF}$ ($\mathcal{ALC}$ plus *functional restrictions*[HS99]), when pronounced.

transitive roles $R \in \mathbf{R}_+$ we define the function $\mathsf{Trans}(R)$ which returns $true$ iff $R \in \mathbf{R}_+$ or $\mathsf{Inv}(R) \in \mathbf{R}_+$. Complex f-$\mathcal{SI}$ concepts are defined by the following syntax rule:

$$C, D \longrightarrow \top \mid \bot \mid A \mid \neg C \mid C \sqcup D \mid C \sqcap D \mid \exists R.C \mid \forall R.C$$

A *terminology*, or $TBox$, is defined by a finite set of *fuzzy concept inclusion* axioms of the form $A \sqsubseteq C$ and *fuzzy concept equalities* of the form $A \equiv C$. Observe that C represents an arbitrary concept, while A an atomic one. This is because dealing with *general terminologies* [HS99] still remains an open problem in fuzzy concept languages.

Let $\mathbf{I} = \{a, b, c, ...\}$ be a set of individual names. A *fuzzy assertion* [Str01] is of the form $\langle a : C \bowtie n \rangle$ or $\langle (a, b) : R \bowtie n \rangle$, where $\bowtie$ stands for $\geq, >, \leq$ and $<$. We call assertions defined by $\geq, >$ *positive* assertions, while those defined by $\leq, <$ *negative* assertions. A finite set of fuzzy assertions defines a fuzzy $ABox$ $\mathcal{A}$. In [Str01] the concept of conjugated pairs of fuzzy assertions has been introduced, in order to represent pairs of assertions that form a contradiction. The possible conjugated pairs are defined in table 5.1, where ϕ represents a concept expression.

	$\langle \phi < m \rangle$	$\langle \phi \leq m \rangle$
$\langle \phi \geq n \rangle$	$n \geq m$	$n > m$
$\langle \phi > n \rangle$	$n \geq m$	$n \geq m$

Table 5.1: Conjugated pairs of fuzzy assertions

A fuzzy set $C \subseteq X$ is defined by its *membership function* (μ_C), which given an object of the universal set X it returns the membership degree of that object to the fuzzy set. By using membership functions we can extend the notion of an *interpretation function*[BMNPS02] to that of a *fuzzy interpretation*. More formally a fuzzy interpretation $\mathcal{I}$ consists of a pair $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is the domain of interpretation, as in the classical case, and $\cdot^{\mathcal{I}}$ is an interpretation function which maps an individual a to an object $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ and a concept A (role R) to a membership function $A^{\mathcal{I}} : \Delta^{\mathcal{I}} \to [0, 1]$ ($R^{\mathcal{I}} : \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \to [0, 1]$), which defines the fuzzy subset $A^{\mathcal{I}}$ ($R^{\mathcal{I}}$) of $\Delta^{\mathcal{I}}$ ($\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$). For example if $a \in \Delta^{\mathcal{I}}$ then $A^{\mathcal{I}}(a)$ gives the degree that the object a belongs to the fuzzy concept A, e.g. $A^{\mathcal{I}}(a) = 0.8$.

In order to extend a fuzzy interpretation to cover arbitrary concepts, created by the syntax rule, we have to interpret the concept forming operators $(\neg, \sqcup, \sqcap, \exists, \forall)$. As in previous approaches to fuzzy DLs [Str01, HKS02, ST04] we use the standard operations to interpret the above concept constructors. These are the Lucasiewicz negation $(c(a) = 1-a)$, the Gödel t-norm $(t(a, b) = min(a, b))$, the Gödel t-conorm $(u(a, b) = max(a, b))$, the Kleen-Dienes fuzzy implication $(\mathcal{J}(a, b) = max(1 - a, b))$ and the supremum and infimum for the existential and universal quantifiers. We refer to the language created by the above operations as f$_{KD}$-$\mathcal{SI}$ after the initials of the name of the fuzzy implication. The semantics of f$_{KD}$-$\mathcal{SI}$ are depicted in table 5.2.

$$\begin{array}{rcl}
\top^{\mathcal{I}}(a) & = & 1 \\
\bot^{\mathcal{I}}(a) & = & 0 \\
(\neg C)^{\mathcal{I}}(a) & = & 1 - C^{\mathcal{I}}(a) \\
(C \sqcup D)^{\mathcal{I}}(a) & = & max(C^{\mathcal{I}}(a), D^{\mathcal{I}}(a)) \\
(C \sqcap D)^{\mathcal{I}}(a) & = & min(C^{\mathcal{I}}(a), D^{\mathcal{I}}(a)) \\
(\forall R.C)^{\mathcal{I}}(a) & = & \inf_{b \in \Delta^{\mathcal{I}}}\{max(1 - R^{\mathcal{I}}(a,b), C^{\mathcal{I}}(b))\} \\
(\exists R.C)^{\mathcal{I}}(a) & = & \sup_{b \in \Delta^{\mathcal{I}}}\{min(R^{\mathcal{I}}(a,b), C^{\mathcal{I}}(b))\} \\
(R^{-})^{\mathcal{I}}(b,a) & = & R^{\mathcal{I}}(a,b) \\
\text{For } R \in \mathbf{R}_{+} & & R^{\mathcal{I}}(a,c) \geq sup_{b \in \Delta^{\mathcal{I}}}\{min(R^{\mathcal{I}}(a,b), R^{\mathcal{I}}(b,c))\}
\end{array}$$

Table 5.2: Semantics of $\mathcal{SI}$-concepts

A fuzzy concept C is *satisfiable* iff there exists some fuzzy interpretation $\mathcal{I}$ for which there is some $a \in \Delta^{\mathcal{I}}$ such that $C^{\mathcal{I}}(a) = n$, and $n \in (0,1]$. A fuzzy interpretation $\mathcal{I}$ satisfies a $TBox$ $\mathcal{T}$ iff $\forall a \in \Delta^{\mathcal{I}} A^{\mathcal{I}}(a) \leq D^{\mathcal{I}}(a)$, for each $A \sqsubseteq C$, and $\forall a \in \Delta^{\mathcal{I}} A^{\mathcal{I}}(a) = D^{\mathcal{I}}(a)$, for each $A \equiv C$.

A fuzzy interpretation satisfies a fuzzy $ABox$ $\mathcal{A}$ if it satisfies all fuzzy assertions in $\mathcal{A}$. In this case, we say $\mathcal{I}$ is a *model* of $\mathcal{A}$. An interpretation $\mathcal{I}$ satisfies a fuzzy assertion if,

$$\begin{array}{rcl}
\langle a : C \geq n \rangle & \text{if} & C^{\mathcal{I}}(a^{\mathcal{I}}) \geq n, \\
\langle a : C \leq n \rangle & \text{if} & C^{\mathcal{I}}(a^{\mathcal{I}}) \leq n, \\
\langle (a,b) : R \geq n \rangle & \text{if} & R^{\mathcal{I}}(a^{\mathcal{I}}, b^{\mathcal{I}}) \geq n, \\
\langle (a,b) : R \leq n \rangle & \text{if} & R^{\mathcal{I}}(a^{\mathcal{I}}, b^{\mathcal{I}}) \leq n.
\end{array}$$

The satisfiability of fuzzy assertions with $>$ and $<$ is defined analogously.

A fuzzy $ABox$ $\mathcal{A}$ is *consistent* iff there exists an interpretation $\mathcal{I}$ that satisfies each fuzzy assertion in the fuzzy $ABox$. We then say that $\mathcal{I}$ is a *model* of $\mathcal{A}$. The *entailment* and *subsumption* problems can be reduced to $ABox$ consistency as shown in [Str01].

5.2 A Fuzzy Tableau for f$_{KD}$-$\mathcal{SI}$

Consistency of an $ABox$ $\mathcal{A}$ can be checked with tableaux algorithms that try to prove the satisfiability of an assertion by constructing a model for it [HST00]. This is accomplished by providing a set of decomposition rules which unfold the possibly complex concept expressions appearing in $\mathcal{A}$. The model is represented by a so-called *completion-forest*, a collection of *completion-trees* some of whose nodes correspond to individuals in the model, each node being labelled with a set of triples of the form $\langle D, \bowtie, n \rangle$ which denote the type, the concept and the membership degree that the individual of the node has been asserted to belong to concept D. As for fuzzy assertions, also when we are dealing with triples of a single node, the concepts of conjugated, positive and negative triples can be defined in the obvious way. Since expansion rules decompose the complex concepts, the concepts that appear in triples are subconcepts of the initial concept. Subconcepts of a concept D are denoted by $sub(D)$. Hence, the set of all subconcepts that appear within an $ABox$ is denoted by $sub(\mathcal{A})$. In the following we assume that all concepts are in their NNF form.

In the present paper we will extend the notions of a *tableau* for an $ABox$ $\mathcal{A}$ [HST00], to a *fuzzy tableau*. In the following we use the symbols $\triangleright$ and $\triangleleft$ as a placeholder for the inequalities $\geq, >$ and $\leq, <$ and the symbol $\bowtie$ as a placeholder for all types of inequations. Furthermore we use the symbols $\bowtie^-$, $\triangleright^-$ and $\triangleleft^-$ to denote their *reflections*. For example the reflection of $\leq$ is $\geq$ and that of $>$ is $<$.

Definition 1 *If $\mathcal{A}$ is an f_{KD}-$\mathcal{SI}$ $ABox$, $\boldsymbol{R}_A$ is the set of roles occurring in $\mathcal{A}$ together with their inverses, $\mathbf{I}_A$ is the set of individuals in $\mathcal{A}$ and $\mathcal{X}$ is the set $\{\geq, >, \leq, <\}$. A fuzzy tableau T for $\mathcal{A}$ is defined to be a quadruple ($\boldsymbol{S}$, $\mathcal{L}$, $\mathcal{E}$, $\mathcal{V}$) such that: $\boldsymbol{S}$ is a set of individuals, $\mathcal{L} : \boldsymbol{S} \to 2^{sub(A)} \times \mathcal{X} \times [0,1]$ maps each individual to a set of triples which denote the membership degree and the type of assertion of each individual to a concept that is a subset of $sub(A)$, $\mathcal{E} : \boldsymbol{R}_A \to 2^{\boldsymbol{S}\times\boldsymbol{S}} \times \mathcal{X} \times [0,1]$ maps each role to a set of triples which denote the membership degree and the type of assertion of a pair of individuals to the role in $\boldsymbol{R}_A$, and $\mathcal{V} : \mathbf{I}_A \to \boldsymbol{S}$ maps individuals occurring in $\mathcal{A}$ to elements in $\boldsymbol{S}$. For all $s, t \in \boldsymbol{S}$, $C, E \in sub(\mathcal{A})$, and $R \in \boldsymbol{R}_A$, T satisfies:*

1. *If $\langle \neg C, \bowtie, n\rangle \in \mathcal{L}(s)$, then $\langle C, \bowtie^-, 1-n\rangle \in \mathcal{L}(s)$,*
2. *If $\langle C \sqcap E, \geq, n\rangle \in \mathcal{L}(s)$ then $\langle C, \geq, n\rangle \in \mathcal{L}(s)$ and $\langle E, \geq, n\rangle \in \mathcal{L}(s)$,*
3. *If $\langle C \sqcup E, \leq, n\rangle \in \mathcal{L}(s)$ then $\langle C, \leq, n\rangle \in \mathcal{L}(s)$ and $\langle E, \leq, n\rangle \in \mathcal{L}(s)$,*
4. *If $\langle C \sqcup E, \geq, n\rangle \in \mathcal{L}(s)$ then $\langle C, \geq, n\rangle \in \mathcal{L}(s)$ or $\langle E, \geq, n\rangle \in \mathcal{L}(s)$,*
5. *If $\langle C \sqcap E, \leq, n\rangle \in \mathcal{L}(s)$ then $\langle C, \leq, n\rangle \in \mathcal{L}(s)$ or $\langle E, \leq, n\rangle \in \mathcal{L}(s)$,*
6. *If $\langle \forall R.C, \geq, n\rangle \in \mathcal{L}(s)$ and there exists a triple $\langle\langle s,t\rangle, \triangleright, n_1\rangle \in \mathcal{E}(R)$ which is conjugated with $\langle\langle s,t\rangle, \leq, 1-n\rangle$ then, $\langle C, \geq, n\rangle \in \mathcal{L}(t)$,*
7. *If $\langle \exists R.C, \leq, n\rangle \in \mathcal{L}(s)$ and there exists a triple $\langle\langle s,t\rangle, \triangleright, n_1\rangle \in \mathcal{E}(R)$ which is conjugated with $\langle\langle s,t\rangle, \leq, n\rangle$ then, $\langle C, \leq, n\rangle \in \mathcal{L}(t)$,*
8. *If $\langle \exists R.C, \geq, n\rangle \in \mathcal{L}(s)$, then there exists $t \in \boldsymbol{S}$ such that $\langle\langle s,t\rangle, \geq, n\rangle \in \mathcal{E}(R)$ and $\langle C, \geq, n\rangle \in \mathcal{L}(t)$,*
9. *If $\langle \forall R.C, \leq, n\rangle \in \mathcal{L}(s)$, then there exists $t \in \boldsymbol{S}$ such that $\langle\langle s,t\rangle, \geq, 1-n\rangle \in \mathcal{E}(R)$ and $\langle C, \leq, n\rangle \in \mathcal{L}(t)$,*

10. *If* $\langle \exists R.C, \leq, n\rangle \in \mathcal{L}(s)$, $\mathsf{Trans}(R)$ *and there exists a triple* $\langle\langle s,t\rangle, \rhd, n_1\rangle \in \mathcal{E}(R)$ *which is conjugated with* $\langle\langle s,t\rangle, \leq, n\rangle$ *then,* $\langle \exists R.C, \leq, n\rangle \in \mathcal{L}(t)$*,*

11. *If* $\langle \forall R.C, \geq, n\rangle \in \mathcal{L}(s)$, $\mathsf{Trans}(R)$ *and there exists a triple* $\langle\langle s,t\rangle, \rhd, n_1\rangle \in \mathcal{E}(R)$ *which is conjugated with* $\langle\langle s,t\rangle, \leq, 1-n\rangle$ *then,* $\langle \forall R.C, \geq, n\rangle \in \mathcal{L}(t)$*,*

12. $\langle\langle s,t\rangle, \bowtie, n\rangle \in \mathcal{E}(R)$ *iff* $\langle\langle t,s\rangle, \bowtie, n\rangle \in \mathcal{E}(\mathsf{Inv}(R))$*,*

13. *There do not exist two conjugated triples in any set of triples for any individual* $x \in$ ***S****,*

14. *If* $\langle a : C \bowtie n\rangle \in \mathcal{A}$*, then* $\langle C, \bowtie, n\rangle \in \mathcal{L}(\mathcal{V}(a))$*,*

15. *If* $\langle (a,b) : R \bowtie n\rangle \in \mathcal{A}$*, then* $\langle\langle \mathcal{V}(a), \mathcal{V}(b)\rangle, \bowtie, n\rangle \in \mathcal{E}(R)$

Analogous properties apply if we substitute $\geq$ by $>$ and $\leq$ by $<$.

Lemma 2 *A fuzzy $\mathcal{SI}$-$ABox$ $\mathcal{A}$ is consistent iff there exists a fuzzy tableau for $\mathcal{A}$.*

5.3 Constructing an f$_{KD}$-$\mathcal{SI}$ Fuzzy Tableau

In order to decide $ABox$ consistency a procedure that constructs a fuzzy tableau for an f$_{KD}$-$\mathcal{SI}$ $ABox$ has to be determined. In the current section we will provide the technical details for constructing a correct tableaux algorithm.

The application of the expansion rules on the concepts of an $ABox$ result in the creation of a *completion-forest*. The nodes of the forest correspond to the individuals that have been generated in order to satisfy positive and negative existential and value restrictions, respectively, and the edges between two nodes, to the relations that connect two individuals. Nodes are labelled with a set of triples $\mathcal{L}(x)$ (*node triples*), which contain concepts that are subsets of $sub(\mathcal{A})$, augmented with the membership degree and the type of assertion that the node belongs to the specific concept. More precisely we define $\mathcal{L}(x) = \{\langle C, \bowtie, n\rangle\}$, where $C \in sub(\mathcal{A})$, $\bowtie\ \in \{\geq, >, \leq, <\}$ and $n \in [0,1]$. Furthermore, edges $\langle x, y\rangle$ are labelled with a set $\mathcal{L}(\langle x,y\rangle)$ (*edge triples*) defined as, $\mathcal{L}(\langle x,y\rangle) = \{\langle R, \bowtie, n\rangle\}$, where $R \in \mathbf{R}_A$. The algorithm expands each tree either by expanding the set $\mathcal{L}(x)$, of a node x with new triples, or by adding new leaf nodes.

If nodes x and y are connected by an edge $\langle x,y\rangle$, then y is called a *successor* of x and x is called a *predecessor* of y, *ancestor* is the transitive closure of *predecessor*. A node x is called an $R-neighbour$ of a node x if either y is a successor of x and $\mathcal{L}(\langle x,y\rangle) = \langle R, \bowtie, n\rangle$ or y is a predecessor of x and $\mathcal{L}(\langle y,x\rangle) = \langle \mathsf{Inv}(R), \bowtie, n\rangle$. We then say that the edge triple *connects* x and y. If we replace $\bowtie$ with $\rhd$ we get the notion of a *positive* R-neighbour and if by $\lhd$ we get that of a *negative* R-neighbour.

Rule	Description	
$(\neg)$	if 1.	$\langle \neg C, \bowtie, n\rangle \in \mathcal{L}(x)$
	2.	and $\langle C, \bowtie^-, 1-n\rangle \notin \mathcal{L}(x)$
	then	$\mathcal{L}(x) \rightarrow \mathcal{L}(x) \cup \{\langle C, \bowtie^-, 1-n\rangle\}$
$(\sqcap_{\rhd})$	if 1.	$\langle C_1 \sqcap C_2, \rhd, n\rangle \mathcal{L}(x)$, x is not indirectly blocked, and
	2.	$\{\langle C_1, \rhd, n\rangle, \langle C_2, \rhd, n\rangle\} \not\subseteq \mathcal{L}(x)$
	then	$\mathcal{L}(x) \rightarrow \mathcal{L}(x) \cup \{\langle C_1, \rhd, n\rangle, \langle C_2, \rhd, n\rangle\}$
$(\sqcup_{\lhd})$	if 1.	$\langle C_1 \sqcup C_2, \lhd, n\rangle \in \mathcal{L}(x)$, x is not indirectly blocked, and
	2.	$\{\langle C_1, \lhd, n\rangle, \langle C_2, \lhd, n\rangle\} \not\subseteq \mathcal{L}(x)$
	then	$\mathcal{L}(x) \rightarrow \mathcal{L}(x) \cup \{\langle C_1, \lhd, n\rangle, \langle C_2, \lhd, n\rangle\}$
$(\sqcup_{\rhd})$	if 1.	$\langle C_1 \sqcup C_2, \rhd, n\rangle \in \mathcal{L}(x)$, x is not indirectly blocked, and
	2.	$\{\langle C_1, \rhd, n\rangle, \langle C_2, \rhd, n\rangle\} \cap \mathcal{L}(x) = \emptyset$
	then	$\mathcal{L}(x) \rightarrow \mathcal{L}(x) \cup \{C\}$ for some $C \in \{\langle C_1, \rhd, n\rangle, \langle C_2, \rhd, n\rangle\}$
$(\sqcap_{\lhd})$	if 1.	$\langle C_1 \sqcap C_2, \lhd, n\rangle \mathcal{L}(x)$, x is not indirectly blocked, and
	2.	$\{\langle C_1, \lhd, n\rangle, \langle C_2, \lhd, n\rangle\} \cap \mathcal{L}(x) = \emptyset$
	then	$\mathcal{L}(x) \rightarrow \mathcal{L}(x) \cup \{C\}$ for some $C \in \{\langle C_1, \lhd, n\rangle, \langle C_2, \lhd, n\rangle\}$
$(\exists_{\rhd})$	if 1.	$\langle \exists R.C, \rhd, n\rangle \in \mathcal{L}(x)$, x is not blocked,
	2.	x has no R-neighbour y connected with a triple $\langle R^*, \rhd, n\rangle$ and $\langle C, \rhd, n\rangle \in \mathcal{L}(y)$
	then	create a new node y with $\mathcal{L}(\langle x, y\rangle) = \{\langle R, \rhd, n\rangle\}$, $\mathcal{L}(y) = \{\langle C, \rhd, n\rangle\}$,
$(\forall_{\lhd})$	if 1.	$\langle \forall R.C, \lhd, n\rangle \in \mathcal{L}(x)$, x is not blocked,
	2.	x has no R-neighbour y connected with a triple $\langle R^*, \rhd, 1-n\rangle$ and $\langle C, \lhd, n\rangle \in \mathcal{L}(y)$
	then	create a new node y with $\mathcal{L}(\langle x, y\rangle) = \{\langle R, \rhd, 1-n\rangle\}$, $\mathcal{L}(y) = \{\langle C, \lhd, n\rangle\}$,
$(\forall_{\rhd})$	if 1.	$\langle \forall R.C, \rhd, n\rangle \in \mathcal{L}(x)$, x is not indirectly blocked, and
	2.	x has a positive R-neighbour y with $\langle C, \rhd, n\rangle \notin \mathcal{L}(y)$ and
	3.	$\langle R^*, \rhd^-, 1-n\rangle$ is conjugated with some edge triple connecting x and y
	then	$\mathcal{L}(y) \rightarrow \mathcal{L}(y) \cup \{\langle C, \rhd, n\rangle\}$,
$(\exists_{\lhd})$	if 1.	$\langle \exists R.C, \lhd, n\rangle \in \mathcal{L}(x)$, x is not indirectly blocked and
	2.	x has a positive R-neighbour y with $\langle C, \lhd, n\rangle \notin \mathcal{L}(y)$ and
	3.	$\langle R^*, \lhd, n\rangle$ is conjugated with some edge triple connecting x and y
	then	$\mathcal{L}(y) \rightarrow \mathcal{L}(y) \cup \{\langle C, \lhd, n\rangle\}$,
$(\forall_+)$	if 1.	$\langle \forall R.C, \rhd, n\rangle \in \mathcal{L}(x)$, Trans (R), x is not indirectly blocked, and
	2.	x has a positive R-neighbour y with $\langle \forall R.C, \rhd, n\rangle \notin \mathcal{L}(y)$ and
	3.	$\langle R^*, \rhd^-, 1-n\rangle$ is conjugated with some edge triple connecting x and y
	then	$\mathcal{L}(y) \rightarrow \mathcal{L}(y) \cup \{\langle \forall R.C, \rhd, n\rangle\}$,
$(\exists_+)$	if 1.	$\langle \exists R.C, \lhd, n\rangle \in \mathcal{L}(x)$, Trans (R), x is not indirectly blocked and
	2.	x has a positive R-neighbour y with $\langle \exists R.C, \lhd, n\rangle \notin \mathcal{L}(y)$ and
	3.	$\langle R^*, \lhd, n\rangle$ is conjugated with some edge triple connecting x and y
	then	$\mathcal{L}(y) \rightarrow \mathcal{L}(y) \cup \{\langle \exists R.C, \lhd, n\rangle\}$,

Table 5.3: Tableaux expansion rules

A node x is *blocked* if for some ancestor y, y is blocked or $\mathcal{L}(x) = \mathcal{L}(y)$. A blocked node x is *indirectly blocked* if its predecessor is blocked, otherwise it is *directly blocked*. If x is directly blocked, it has a unique ancestor y that blocks it.

The algorithm initializes a forest $\mathcal{F}_{\mathcal{A}}$ to contain a root node x_0^i, for each individual $a_i \in \mathbf{I}$ occurring in the $ABox$ $\mathcal{A}$ and additionally $\{\langle C_i, \bowtie, n\rangle\} \cup \mathcal{L}(x_0^i)$, for each assertion of the form $\langle a_i : C_i \bowtie n\rangle$ in $\mathcal{A}$, and an edge $\langle x_0^i, x_0^j\rangle$ if $\mathcal{A}$ contains an assertion $\langle (a_i, a_j) : R_i \bowtie n\rangle$, with $\{\langle R_i, \bowtie, n\rangle\} \cup \mathcal{L}(\langle x_0^i, x_0^j\rangle)$ for each assertion of the form $\langle (a_i, a_j) : R_i \bowtie n\rangle$ in $\mathcal{A}$. $\mathcal{F}_{\mathcal{A}}$ is then expanded by repeatedly applying the rules from table 5.3. We use the notation R^* to denote either the role R or the role returned by $\mathsf{Inv}(R)$. Observe in table 5.3 if a value or existential restriction in a node x are to be propagated to a node y, the membership degrees of the propagated concepts in y would be the same as the ones in node x. The proof of this property is a quite technical one and it is omitted from here.

In description logics the notion of a *clash* is used in order to denote that a contradiction has occurred in the completion forest. In our framework a node x is said to contain a *clash* if and only if there exist two conjugated triples within a single node, or one of the following triples exists within a node:

$$\begin{array}{l} \langle \bot, \geq, n\rangle, \langle \top, \leq, n\rangle, \text{ for } n > 0,\ n < 1 \text{ respectively} \\ \langle \bot, >, n\rangle, \langle \top, <, n\rangle \\ \langle C, <, 0\rangle, \langle C, >, 1\rangle \end{array}$$

Lemma 3 *Let $\mathcal{A}$ be an f$_{KD}$-$\mathcal{SI}$ $ABox$. Then*

1. *The tableaux algorithm terminates*

2. *$\mathcal{A}$ has a tableau if and only if the expansion rules can be applied to $\mathcal{A}$ such that they yield a complete and clash-free completion forest.*

Theorem 4 *The tableaux algorithm is a decision procedure for the consistency of f-$\mathcal{SI}$ $ABoxes$ and the satisfiability and subsumption of f-$\mathcal{SI}$ concepts with respect to* simple *terminologies.*

Theorem 4 is an immediate consequence of lemma 3. Moreover, in [Str01], it was proved that subsumption of fuzzy concepts can be reduced to $ABox$ satisfiability. More precisely, $C \sqsubseteq D$ iff $\mathcal{A} = \{\langle a : C \geq n\rangle, \langle a : D < n\rangle\}$, with $n \in \{n_1, n_2\}$, $n_1 \in (0, 0.5]$ and $n_2 \in (0.5, 1]$, is unsatisfiable.

5.3.1 Example

To see the expressive power of the extended language lets consider an example. Suppose we have the fuzzy $ABox$:

$$\begin{aligned}\mathcal{A}= \quad &\{\langle (PLANE, WING) : hasBigPart \geq 0.7\rangle,\\ &\langle (WING, ENGINE) : hasBigPart \geq 0.8\rangle,\\ &\langle (ENGINE, COOLINGSYSTEM) : hasBigPart \geq 0.6\rangle,\\ &\langle COOLINGSYSTEM : Faulty \geq 0.6\rangle\}\end{aligned}$$

and $RBox$, $\mathcal{R} = \{(hasBigPart)\}$. We want to know if $PLANE$ has a a big faulty part at a degree greater than 0.6. In order to answer, we test the consistency of the system $\mathcal{A} \cup \{\langle PLANE : \exists hasBigPart.Faulty < 0.6\rangle\}$. First, we initialize a completion-forest, as described in section 5.3 and then apply the expansion rules of table 5.3. Cause of the $\exists_+$ rule the existential restrictions would be propagated along the path and thus we would have $\langle \exists hasBigPart.Faulty, <, 0.6\rangle \in \mathcal{L}(ENGINE)$. Then, rule $\exists_<$ would add $\langle Faulty, <, 0.6\rangle \in \mathcal{L}(COOLINGSYSTEM)$ which causes a clash with the triple $\langle Faulty, \geq, 0.6\rangle \in \mathcal{L}(COOLINGSYSTEM)$. So we can conclude that there is a faulty big part at a degree greater or equal than 0.6.

5.4 Reduction to Crisp $\mathcal{SI}$

In [Str04] a reduction of f$_{KD}$-$\mathcal{ALC}$ to classical (crisp) $\mathcal{ALC}$ was provided. The purpose of this reduction is to reduce the reasoning problem from the fuzzy $\mathcal{ALC}$ language to the crisp $\mathcal{ALC}$ and use existing optimized reasoners, like FaCT [Hor98c] to perform inference services. Apart from using the optimized reasoners that exist for classical DLs this approach additionally solves the problem of general terminologies [Baa90]. Such a reduction is also possible in the case of f$_{KD}$-$\mathcal{SI}$. For example, in [Str04], the fuzzy assertion $\langle a : \forall R.C \geq 0.7\rangle$ is transformed to the crisp assertion $a : \forall R_{>0.3}.C_{\geq 0.7}$, where $R_{>0.3}$ ($C_{\geq 0.7}$) is a newly introduced crisp role (concept) in the system. This means that if a pair $(a, b) : R_{>0.3}$ exists the classical rule for propagating value restrictions [HST00] would add $\forall R_{>0.3}.C_{\geq 0.7}$ to b, which is correct according to what was said in section 5.3.

In [Str04], for a fuzzy role R that appears in the fuzzy KB, and for two membership degrees $c_1, c_2 \in [0, 1]$, with $c_1 \geq c_2$, that appear in a fuzzy $ABox$ $\mathcal{A}$, two new terminological axiom of the form $R_{\geq c_1} \sqsubseteq R_{>c_2}$ and $R_{>c_2} \sqsubseteq R_{\geq c_2}$ were added in the crisp KB in order for the classical deduction algorithm to derive correct conclusion for the f$_{KD}$-$\mathcal{ALC}$. To complete the reduction, in the presence of transitive role axioms, if $\mathsf{Trans}(R)$ appears in the fuzzy $RBox$, then for each new role $R_{\bowtie c}$, with $c \in [0, 1]$, we need a new transitive role axioms in the crisp $RBox$ of the form $\mathsf{Trans}(R_{\bowtie c})$. Observe that the reduced language is actually $\mathcal{SHI}$ ($\mathcal{ALCH}$ in the case of the reduction of f$_{KD}$-$\mathcal{ALC}$), where $\mathcal{H}$ denotes role hierarchies [HS99], and not $\mathcal{SI}$ (respectively $\mathcal{ALC}$) [Str04]. It is well known [HS99] that the presence of role hierarchies makes DL logics EXPTIME-hard. Practically good behavior for such logics [Hor98c] is due to the fact that realistic knowledge bases contain few role inclusion axioms. But the reduction of f$_{KD}$-$\mathcal{SI}$ would create up to $2|\mathcal{R}|(|\mathcal{N}|-1)$ role inclusion axioms, where $\mathcal{R}$ and $\mathcal{N}$ are the number of different roles and membership degrees appearing in the fuzzy KB (plus degrees 0, 0.5 and 1) [Str04], respectively. For example for 10 different roles and membership degrees we would have about 200 role inclusion axioms, an unnecessarily big number. Moreover, if in the future we consider the extension of languages such as $\mathcal{SHI}$ with fuzzy set theory, this number can become even greater. Yet, we have no practical results about reasoning with classical DL reasoners in such reduced languages.

Since fuzzy DLs that use the min-max operations and the Kleene-Dienes fuzzy implication are directly compatible to classical DLs, there would also be subject to the same or similar optimization methods, used in DL reasoners [Hor98c].

Chapter 6

Semantics Driven Support for Query Formulation

In this section we describe the principles of the design and development of an intelligent query interface system; which, in the rest of the section, will be simply called *query tool*. This system is meant to support a user in formulating a precise query – which best captures the user information needs – even in the case of complete ignorance of the vocabulary of the underlying information system holding the data. The final purpose of the tool is to generate a conjunctive query ready to be executed by some evaluation engine associated to the information system.

6.1 Introduction

The intelligence of the interface is driven by an ontology describing the domain of the data in the information system. The user can exploit the ontology's vocabulary to formulate the query, and she/he is guided by such a rich vocabulary in order to understand how to express her/his information needs more precisely, given the knowledge of the system. This latter task – called *intensional navigation* – is the most innovative functional aspect of our proposal. Intensional navigation can help a less skilled user during the initial step of query formulation, thus overcoming problems related with the lack of schema comprehension and so enabling her/him to easily formulate meaningful queries. The user may specify her/his request using generic terms, refine some terms of the query or introduce new terms, and iterate the process. Moreover, users may explore and discover general information about the domain without querying the information system, giving instead an explicit meaning to a query and to its subparts through classification.

In the literature there are several approaches at providing intelligent visual query systems for relational or object oriented databases (see [CCLB97] for an extensive survey). However, to our knowledge, the work presented in this section is among the first well-founded intelligent systems for query formulation support in the context of ontology-based query processing. The strength of our approach derives from the fact that the graphical and natural language representation of the queries is underpinned by a formal semantics provided by an ontology language. The use of an appropriate ontology language enables the system engineers to precisely describe the data sources, and their implicit data constraints, by means of a system global ontology (see [CGL$^+$98]). The same ontology is leveraged by the query interface to support the user in the composition of the query, rather than relying on a less expressive logical schema. The underlying technology used by the query interface is based on the recent work on query containment under constraints (see [CDGL98, HSTT00]).

6.2 Query interface: the user perspective

Initially the user is presented with a choice of different query scenarios which provide a meaningful starting point for the query construction. The interface guides the user in the construction of a query by means of a diagrammatic interface, which enables the generation of precise and unambiguous query expressions. Moreover, interface presentation and behaviour are entirely guided by the ontology. This is achieved by leveraging the correct and complete automatic reasoning on the ontology language (see Section 6.3).

Query expressions are compositional, and their logical structure is not flat but tree shaped; i.e. a node with an arbitrary number of branches connecting to other nodes. This structure corresponds to the natural linguistic concepts of noun phrases with one or more propositional phrases. The latter can contain nested noun phrases themselves.

A query is composed by a list of terms coming from the ontology (classes); e.g. “Supplier” and “Multinational”. Branches are constituted by a property (attributes or associations) with its value restriction, which is a query expression itself; e.g. “selling on Italian market”, where “selling on” is an association, and “Italian market” is an ontology term.

The focus paradigm is central to the interface user experience: manipulation of the query is always restricted to a well defined, and visually delimited, subpart of the whole query (the *focus*). The compositional nature of the query language induces a natural navigation mechanism for moving the focus across the query expression (nodes of the corresponding tree). A constant feedback of the focus is provided on the interface by means of the kind of operations which are allowed. The system suggests only the operations which are “compatible” with the current query expression; in the sense that do not cause the query to be unsatisfiable. This is verified against the formal model describing the data sources.

One of the main requirements for the interface is that it must be accessed by any HTML browser, even in presence of restrictive firewalls. This constraints the its design, which overall appearance is shown in Figure 6.1.

The interface is composed by three functional elements. The first one (top part) shows a representation of the query being composed, and the current focus. The second one is the query manipulation pane (bottom part) containing tools to specialise the query. The third component is the result pane containing a table which presents the actual results. The first two components are used to compose the query, while the third one is used to specify the data which should be retrieved from the data sources. We concentrate on the query building part; therefore we wont discuss the query result pane, which allows the user to define the columns of a table which is going to organise the data from the query result.

Query textual representation The first component consists of a tree representing the query expression in a natural language fashion. The user selects subparts of the query for further refinement. The selection defines the current focus, which will be represented in the diagrams described in the following sections. The selected subexpression can be modified (refined or extended) by means of the query manipulation pane. When a node is selected, then the system automatically selects the whole subtree rooted at the node selected by the user.

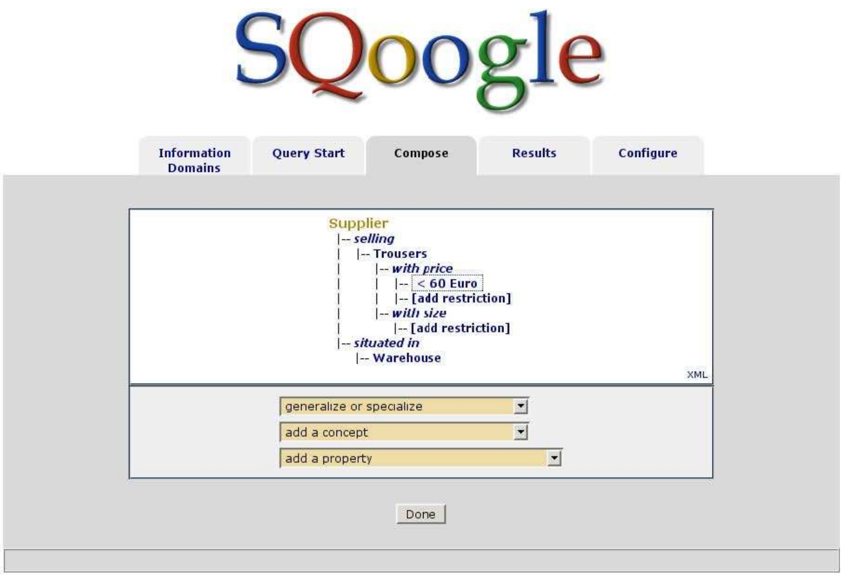
It is important to stress that, although natural language is used as feedback to represent the query, this is used in generation mode only. Since the user does not write queries directly, there is no need to parse any natural language sentence or to resolve linguistic ambiguities.

Query manipulation pane The elements in the pane represent the current selection, and the operations allowed in its context. It is organised as a set of pop-up menus enabling the refinement of the current focus. There are several menus corresponding to different operations; however, the interface shows only the menus for the operations which make sense w.r.t. the current query.

The first menu, labeled as generalize or specialize, enables what we call *substitution by navigation*; i.e. the possibility of substituting the selected portion of the query with a more specific or more general terms. The proposed terms in the menu are either more specific or more general w.r.t. the query expression *from the focus viewpoint*.By selecting one of these terms, the user can substitute the whole focus with the selected term. The purpose of the substitution group is twofold: it enables the replacement of the focus and it shows the position of the selection w.r.t. the terms in the ontology.

It can be the case that in the ontology there are terms which are equivalent to the selected part. In this case the user is offered to replace the selection with the equivalent term by the activation of the Replace Equivalent button.

A different refinement enabled by the interface is by *compatible terms*. These are terms in the ontology whose overlap with the focus can be non-empty. These ontology terms can be added to the head of the selection by using the add a concept pop-up menu. For example, “Student” is among the compatible terms for the focus “Employee”,



```
Supplier
 |-- selling
 |    |-- Trousers
 |         |-- with price
 |         |    |-- < 60 Euro
 |         |    |-- [add restriction]
 |         |-- with size
 |              |-- [add restriction]
 |-- situated in
      |-- Warehouse
```

Figure 6.1: Query building interface.

but “Textile” is not. The compatible terms are automatically suggested to the user by means of appropriate reasoning task on the ontology describing the data sources.

Analogously, the user can add properties to the focus: *associations* (e.g. “Industry with sector”), and/or *attributes* (e.g. “Employee whose name is”). The main difference between the two kinds of property lies on the class representing the range of the property. In the attribute case it is a basic data type like “String” or “Integer”, while for an association it is a generic class. This can be performed by means of a `add a property` pop-up menu, which presents the possible alternatives. Name and value restrictions for each property are verbalised using meta information associated to the terms in the ontology. For example, the association “with sector” with the restriction “Textile” is shown as “belonging to the textile sector”.

Note that the terms and the properties proposed by the system depend on the overall query expression, not only on the focus. This means that subparts of the query expression, taken in isolation, would generate different suggestions w.r.t. those in their actual context in the query.

6.3 Query interface: inside the box

In this section we describe the underpinning technologies and techniques enabling the user interface described in the previous sections. We will start by describing our assumptions on the query language, followed by system perspective over the described query building process. The whole system is supported by formally defined reasoning services which are described in Section 6.3.2. Finally, we introduce the verbalisation mechanism which enables the system to show the queries in a natural language fashion.

6.3.1 Conjunctive queries

Since the interface is build around the concept of classes and their properties, we consider conjunctive queries composed by unary (classes) and binary (attribute and associations) terms.

The body of a query can be considered as a graph in which variables (and constants) are nodes, and binary terms are edges. A query is connected (or acyclic) when for the corresponding graph the same property holds. Given the form of query expressions composed by the interface introduced in Section 6.2, we restrict ourselves to acyclic connected queries. This restriction is dictated by the requirement that the casual user must be comfortable with the language itself.[1] Note that the query language restrictions do not affect the ontology language, where the terms are defined by a different (in our case more expressive) language. The complexity of the ontology language is left completely hidden to the user, who doesn't need to know anything about it.

To transform any query expression in a conjunctive query we proceed in a recursive fashion starting from the top level, and transforming each branch. A new variable is associated to each node: the list of ontology terms corresponds to the list of unary terms. For each branch, it is then added the binary query term corresponding to the property, and its restriction is recursively expanded in the same way.

Let us consider for example the query "Supplier and Multinational corporation selling on Italian market located in Europe", with the meaning that the supplier is located in Europe. Firstly, a new variable (x_1) is associated to the top level "Supplier and Multinational corporation". Assuming that the top level variable is by default part of the distinguished variables, the conjunctive query becomes

$$\{x_1 \mid \text{Suppl}(x_1), \text{Mult\_corp}(x_1), \ldots\},$$

where the dots mean that there is still part of the query to be expanded. Then we consider the property "selling on", with its value restriction "Italian market": this introduces a new variable $x_{1,1}$. The second branch is expanded in the same way generating the conjunctive query

$$\{x_1 \mid \text{Suppl}(x_1), \text{Mult\_corp}(x_1), \text{sell\_on}(x_1, x_{1,1}), \text{It\_market}(x_{1,1}), \text{loc\_in}(x_1, x_{1,2}), \text{Eur}(x_{1,2})\}.$$

This transformation is bidirectional, so that a connected acyclic conjunctive query can be represented as a query expression (in the sense of Section 6.2) by dropping the variable names.

[1] Our technique can deal with disjunction of conjunctive queries, even with a limited form of negation applied to single terms. See [CDGL98, HSTT00] for the technical details.

Since a query is a tree, the focus corresponds to a selected sub-tree. It is easy to realise that each sub-tree is univocally identified by the variable corresponding to a node. Therefore, the focus is always on variable, and moving the focus corresponds to selecting a different variable. Modifying a query sub-part means operating on the corresponding sub-tree modifying the corresponding query tree.

Substitution by navigation corresponds to substitute the whole sub-tree with the chosen ontology term. The result would be a tree composed by a single node, without any branch, whose unary term is the given ontology term. In the *refinement by compatible terms*, the selected terms are simply added to the root node as unary query terms. For the *property extension*, adding an attribute or associations corresponds to the creation of a new branch. This operation introduces a new variable (i.e. node) with the corresponding restriction. When an attribute is selected, and a constant (or an expression) is specified, then this is added as restriction for the value of the variable.

6.3.2 Reasoning services and query interface

Reasoning services w.r.t. the ontology are used by the system to drive the query interface. In particular, they are used to discover the terms and properties (with their restrictions) which are proposed to the user to manipulate the query.

Our aim is to be as less restrictive as possible on the requirements for the ontology language. In this way, the same technology can be adopted for different frameworks, while the user is never exposed to the complexity (and peculiarities) of a particular ontology language.

In our context, an ontology is composed by a *set of predicates* (unary, binary), together with a *set of constraints* restricting the set of valid interpretations (i.e. databases) for the predicates. The kind of constraints which can be expressed defines the expressiveness of the ontology language. Note that these assumptions are general enough to take account of widely used modelling formalisms, like OWL-DL or UML.

We do not impose general restrictions on the expressiveness of the ontology language; however, we require the availability of two *decidable* reasoning services: satisfiability of a *conjunctive query*, and containment test of two conjunctive queries, both w.r.t. the constraints. If the query language includes the *empty* query (i.e. a query whose extension is always empty), then query containment is enough (a query is satisfiable iff it is not contained in the empty query). As described in Section 6.2, the query building interface represents the available operations on the query w.r.t. the current focus; i.e. the variable which is currently selected. Therefore, we need a way of describing a conjunctive query from the point of view of a single variable. The expression describing such a viewpoint is still a conjunctive query; which we call *focused*. This new query is equal to the original one, with the exception of the distinguished (i.e. free) variables: the only distinguished variable of the focused query is the variable representing the focus. In the following we represent as q^x the query q focused on the variable x. For example, the query

$$q \equiv \{x_1, x_{1,2} \,|\, \text{Mult\_corp}(x_1), \text{sell\_on}(x_1, x_{1,1}), \text{It\_market}(x_{1,1}), \text{loc\_in}(x_1, x_{1,2}), \text{Eur}(x_{1,2})\},$$

focused in the variable $x_{1,1}$ would simply be

$$q^{x_{1,1}} \equiv \{x_{1,1} \,|\, \text{Mult\_corp}(x_1), \text{sell\_on}(x_1, x_{1,1}), \text{It\_market}(x_{1,1}), \text{loc\_in}(x_1, x_{1,2}), \text{Eur}(x_{1,2})\}.$$

The operations on the query expression require two different types of information: *hierarchical* (e.g. substitution by navigation), and on *compatibility* (e.g. refinement and new properties).

Let us consider the substitution by navigation with the more specific terms (the cases with more general and equivalent terms are analogous). Given the focused query q^x, we are interested to the unary atomic terms T s.t. the query $\{y \,|\, T(y)\}$ is contained in q^x and it is most general (i.e. there is no other query of that form contained in q^x, and containing $\{y \,|\, T(y)\}$).

Refinement by compatible terms and the addition of a new property to the query require the list of terms "compatible" with the given query. In terms of conjunctive queries, this corresponds to add a new term to the query. The term to be added should "join" with the query by means of the focused variable, and must be compatible in the sense that the resulting query should be satisfiable. This leads to the use of satisfiability reasoning service to check which predicates in the ontology are compatible with the current focus. With unary terms this check corresponds simply to the addition of the term $T(x)$ to the focused query q^x, and verify that the resulting query is satisfiable.

The addition of a property requires the discovery of both a binary term and its restriction: the terms to be added are of the form $\{x \,|\, R(x,y), T(y)\}$ if the focused variable is x. As for the refinement by compatible terms, the system should check all the different binary predicates from the ontology for their compatibility. This is practically performed by verifying the satisfiability of the query $q^x \bowtie \{x \,|\, R(x,y)\}$, for all atomic binary predicates R in the signature and where y is a variable not appearing in q.[2] Once a binary predicate R is found to be compatible with the focused query, the restriction is selected as the most general unary predicate T such that the query $q^x \bowtie \{x \,|\, R(x,y), T(y)\}$ is satisfiable.

6.3.3 Using a Description Logics Reasoner

Although our approach is not tight to any ontology language, in the test implementation of our system we are using Description Logics (DLs). The reasons for this choice lie in the facts that DLs can capture a wide range of widespread modelling frameworks, and the availability of efficient and complete DL reasoners.

[2]Here $\bowtie$ represents a natural join.

We adopted the Description Logics $\mathcal{SHIQ}$ (see [HS02]); which is expressive enough for our purposes, and for which there are state of the art reasoners. Note that the adoption of $\mathcal{SHIQ}$ allow us to use ontologies written in standard Web Ontology languages like OWL–DL (see [HPS03b]).

For space limitations we are not going to describe in detail the underlying $\mathcal{SHIQ}$ DL; the reader is referred to the above mentioned bibliographic references. The ontology contains unary (concepts) and binary (roles) predicates, and the constraints are expressed by means of inclusion axioms between concept or role expressions. One of the key features of $\mathcal{SHIQ}$ is the possibility of expressing the inverse of a role; which is extremely useful for converting tree–shaped queries into DL concept expressions.

Given the restriction to tree–shaped conjunctive query expressions, together with the availability of inverse roles, a focused query (see Section 6.3.2) corresponds to a concept expression (see [HT02]). Therefore, all the reasoning tasks described in Section 6.3.2 correspond to standard DL reasoning services. Again, this is not a restriction imposed by the underlying technology, since general conjunctive queries can be dealt with techniques described in [CDGL98, HSTT00].

The idea behind the transformation of a query expression into a single concept description is very simple, and it is based on the fact that a concept expression can be seen as a query with a single distinguished variable. To focus the query on a variable, we start from the variable itself, then we traverse the query graph by encoding binary terms into DL existential restrictions and dropping the variable names. The fact that queries are tree–shaped ensures that variable names can be safely ignored. Let us consider for example the query expression

$$\{\text{Mult\_corp}(x_1), \text{Italian}(x_1), \text{sell\_on}(x_1, x_{1,1}), \text{It\_market}(x_{1,1})\}.$$

The DL expression corresponding to the query focused on $x_{1,1}$ is

$$(\text{It\_market} \sqcap \exists \text{sell\_on}^{-}(\text{Mult\_corp} \sqcap \text{Italian}));$$

where sell_on^{-} corresponds to the inverse of sell_on role.

As explained in Section 6.3.2, we need two kinds of information: hierarchical and compatibility. These, in the DL framework, are provided by the standard reasoning services of satisfiability and taxonomy position of a concept expression respectively. The first service verifies the satisfiability w.r.t. a knowledge base; while the second classifies a concept expression (i.e., provides it w.r.t. the ISA taxonomy of concept names).[3] Reasoning tasks described in Section 6.3.2 can be straightforwardly mapped into satisfiability and classification.

For example, checking the compatibility of the term Italian with the query

$$\{\text{Mult\_corp}(x_1), \text{sell\_on}(x_1, x_{1,1}), \text{It\_market}(x_{1,1})\},$$

[3]DL systems usually provide an efficient way of obtaining the taxonomic position of a given concept expression.

is performed by checking the satisfiability of the concept

$$\text{Italian} \sqcap \text{Mult\_corp} \sqcap \exists \text{sell\_on} \text{It\_market}.$$

Compatibility of binary terms is performed analogously by using an existential restriction; e.g., $\exists \text{sell\_on} \top$.[4] To discover the restriction of a property we use classification instead of repeated satisfiability. The idea is to classify the query focused on the variable introduced by the property. For example, to discover the restriction of sell_on applied to the query expression

$$\{x_1 \,|\, \text{Mult\_corp}(x_1), \text{Italian}(x_1)\},$$

we classify the expression $\exists \text{sell\_on}^{-}(\text{Mult\_corp} \sqcap \text{Italian}))$. The DL reasoner returns the list of concept names more general and equivalent to the range of the relation sell_on, when restricted to the domain $(\text{Mult\_corp} \sqcap \text{Italian})$. This is exactly the information we need to discover the least general predicate(s) which can be applied to the property in the given context.

Our implementation uses the DL reasoner Racer (see [HM01c]); which fully supports the $\mathcal{SHIQ}$ DL. The interaction with the DL reasoner is based on the DIG 1.0 interface API (see [BMC03a]), a standard to communicate with DL reasoners developed among different DL systems implementors. This choice makes our system independent from a particular DL reasoner, which can be substituted with any DIG based one.

6.3.4 Query verbalisation

The system always presents the user with a natural language transliteration of the terms in conjunctive query. This is performed in an automatic way by using meta information associated with the ontology terms, both classes and properties. The verbalisation of the ontology terms must be provided in advance by the ontology engineers. For the verbalisation we use an approach similar to the one adopted by the Object Role Modelling framework (ORM, see [Hal01, ORM03]).

Each class name in the ontology has associated a short noun phrase (usually one or two words), which represents the term in a natural language fashion. For example, to the class *PStudent* is associated "Postgraduate student" The user will see only the associated sentence, while *PStudent* is just used in the internal ontology representation.

For (binary) associations the ontology engineer has to provide two different verbalisations for the two directions. For example, let assume that the ontology states that the association *occ_room* links the two classes *PStudent* and *Room*. Then the engineer associates to the association the verbalisation "occupies" for the direction from *PStudent* to *Room*, and the verbalisation "is occupied by" for the other direction.

[4]Note the use of the $\top$ concept representing the whole domain (any possible concept).

Attributes need one direction only, since they are never used from the point of view of the basic data type. In this case, the engineer is only required to provide the attribute verbalisation from the point of view of the class.

6.4 Discussion

The work here described deals with a relatively new problem, namely providing the user with a visual interface to query heterogeneous data sources through an integrated ontology (that is, a set of constraints), and a specific literature does not exist yet. By looking at the extensive survey on Visual Query System (VQS) presented in [CCLB97] it easy to see that only little work has been done in the specific context we are dealing with. Some preliminary work was done by one research group [BF96, Fra00, BNP00, BF02]. Similar work, from the point of view of the visual interface paradigm, was carried out in the context of the Tambis project [MGP98, BSN$^+$99]; however that system lacked of the well founded support provided by a logic-based semantics. Also [BMR99] contains some interesting approach from the point of view of the visual interface, but again the system has a different background semantics.

In fact, only recently research has started to have a serious interest in query processing and information access supported by ontologies. Recent work has come up with proper semantics and with advanced reasoning techniques for query evaluation and rewriting using views under the constraints given by the ontology – also called view-based query processing [Ull97, CGL00]. This means that the notion of accessing information through the navigation of an ontology modelling the information domain has its formal foundations.

This paper has presented the first well-founded intelligent user interface for query formulation support in the context of ontology-based query processing. This paper hopefully proved that our work has been done in a rigorous way both at the level of interface design and at the level of ontology-based support with latest generation logic-based ontology languages such as description logics, DAML+OIL and OWL. However, there are open problems and refinements which have still to be considered in our future work.

The system uses the verbalisations described in Section 6.3.4 to transform the conjunctive query into a natural language expression closer to the user understanding. In the course of the SEWASIE project some effort will be dedicated to explore semi-automatic techniques to rephrase the expressions in more succinct ways without loosing their semantic structure.

Another important aspect to be worked out is the understanding of the effective methodologies for query formulation in the framework of this tool, a task that needs a strong cooperation of the users in its validation. This task is proceeds in parallel with the interface user evaluation.[5] The other crucial aspect is the efficiency and the scalability of

[5]An on-line prototypical version of the query building tool is available at the URL `http://frida.`

the ontology reasoning for queries. We are currently experimenting the tool with various ontologies in order to identify possible bottlenecks.

`inf.unibz.it/sewasie`.

Chapter 7

Approximating Terminological Queries

This chapter elaborates an implementation issues related to approximating Semantic Web inferences. For this purpose this chapter focuses on the algorithms described and evaluated in D2.1.2 “Methods for Approximate Reasoning” and reports some implementation details.

In general there are three possibilities for approximating Semantic Web inferences (see also D2.1.2 “Methods for Approximate Reasoning”):

Language Weakening where the representation formalism is simplified by omitting some constructs.

Knowledge Compilation where the ontology is translated into another representation formalism. During the translation the knowledge can be simplified or some information can be lost.

Approximate Deduction The inference itself is approximated.

From an implementation point of view Language Weakening and Knowledge Compilation can be handled by the same principle. A brief overview will be given in Section 7.1 and 7.2. The following Section 7.3 discusses the implementation of Approximate Deduction.

7.1 On Implementing Language Weakening

Trough Language Weakening an ontology can be simplified by omitting some parts, e.g. all sub terms consisting of a negation such as $\neg\Phi$. Therefore Language Weakening can be done a priori. Before any query will be answered the ontology can be transferred into a simplified one where some constructors are omitted. The translator can be seen as a black box which takes an ontology as input and returns the simplified ontology as output.

Given the simplified ontology an appropriate reasoner can answer the queries. In the case of Language Weakening the reasoner may be specialized for the simplified ontology where, now, some constructs need not be considered but its inference principle will not be changed.

Apart from the translation of the ontology it may also be required to translate the query in an appropriate query format. The query has to be reformulated with respect to the translation of the ontology itself and the new representation formalism of the simplified ontology. The query translation should reflect the assumptions and simplifications which were made during the ontology translation. Therefore the translator should be able to translate the query into the format which meets the requirements of the simplified ontology and its intended reasoning methods.

7.2 On Implementing Knowledge Compilation

Knowledge Compilation aims at pre-processing the ontology off-line such that on-line reasoning becomes faster and can also be done a priori. Moreover from an implementation point of view it can be seen as an extension of implementing Language Weakening. A translator translates ontology and query into a new target format. In contrast to Language Weakening the target format may be different from the source format. Therefore the reasoner may be replaced completely by a new one which is able to reason with the new format. For example, an OWL DL ontology can be translated into disjunctive logic programs which can be handled by resolution[1] (see D2.1.2 "Methods for Approximate Reasoning" for more details).

The translator may use further services during its compilation. For example it can use a description logic reasoner. The DL reasoner determines the subsumption hierarchy which the translator extracts and includes in the new simplified ontology. During query answering the subsumption hierarchy need not be recalculated. In other words the compilation makes implicit knowledge explicit. For Knowledge Compilation also an additional task must be handled by the translator. Because query and ontology are in a different format the answers to a query will also be in different format. Therefore it may also be necessary to translate back the results to the original format.

To summarize, Language Weakening and Knowledge Compilation can be implemented using a translator which translates ontology, queries and their respective answers from source format into target format and back.

[1]The approach is currently under development.

7.3 Implementing Approximate Reasoning

Approximate reasoning concentrates on the dynamic part only when a concrete query is sent to the system which uses approximate reasoning to answer that query. The design of such an approximated inference engine will be the focus of this section.

The first design decision during the development of an approximated inference engine is: should the approximated inference engine completely replace the normal engine or should the approximated inference engine use the normal engine as much as possible.

The complete replacement has the same properties as a translator because it can be seen as a black box which receives a query as input and returns the answers as output. However it ignores all the improvements and optimizations which was made in the last years during the development of normal reasoners, i.e. description logic reasoners like Fact [Hor98a],Fact++ (see chapter 3) or Racer [HM01a]. The charming alternative uses the normal description logic reasoner as much as possible and can profit from the improvements and optimizations in a description logic reasoner.

Approximation with using a description logic reasoner is possible because most inferences in many description logics can be reduced to satisfiability tests with respect to an ontology. For example, the subsumption check, i.e. whether a concept expression C is more specific than D ($C \sqsubseteq D$), can be reformulated as a satisfiability test where $C \sqcap \neg D$ is checked for unsatisfiability. The more complex inference services like classification or instance retrieval can be reduced to a sequence of satisfiability/subsumption test. Inferences of a normal description logic reasoner can be divided into two groups: the basic inference, i.e. the satisfiability check[2], and the set of complex inferences which can be reduced to the basic inference.

An approximated inference method can exploit this dichotomy of inferences. It can approximate the complex inferences by adopting the reduction to the basic inference. This means that approximation changes the generation of the sequence of satisfiability tests. For example an approximated classification generates a different sequence of satisfiability test as the normal classification. The satisfiability check itself can be performed by a normal DL reasoner; only the generation of the sequence will be changed. Therefore, optimization improvements for the complex inferences may be lost or may be adopted by the approximated complex inference, but the improvements for performing each individual satisfiability test are still usable.

The approximated sequence can differ from the normal sequence by the order in which satisfiability tests will be performed. For example, the Cadoli-Schaerf approximations introduce a number of simplified satisfiability tests for each normal satisfiability test. The simplified tests can be performed more efficiently and avoid some expensive normal tests (see D2.1.2 “Methods for Approximate Reasoning” for more details).

[2]The subsumption check can also serve as a basic inference where the compound inferences can be reduced to. For simplicity we use the satisfiability check.

Apart from the complex inferences also the basic inference can be approximated. The engine which normally performs the satisfiability test can be replaced by an approximated one. More formally, for an expression Φ the approximated inference engine $\mid\!\sim_\xi$ checks the satisfiability with respect to an ontology/TBox $\mathcal{T}$, i.e. $\mathcal{T} \mid\!\sim_\xi \Phi$. ξ indicates which approximation method is used.

The approximated inference engine — as every normal reasoner — must analyze Φ. Even for description logics the complexity of reasoning depends on the language used to represent Φ. More precisely, the operators used in the definitions of terms such as Φ determine the complexity of the reasoning. If the approximated engine omits parts of Φ or at least approximate the reasoning needed for some operators then the reasoning performance can possibly be improved. However, the same effect of improving reasoning performance can be achieved by simplifying Φ with the function ξ to $\xi(\Phi)$ and then using normal inferences. For example if the approximated reasoner $\mid\!\sim_\xi$ omits some parts in Φ the same parts can be omitted in $\xi(\Phi)$. Approximating the reasoning for given operators function ξ can simulate the approximation by simplifying the operators in Φ. In other word, there is a strong correlation between many approximated engines $\mid\!\sim_\xi$ and a simplified expression ξ for normal inferences $\mid\!-$. Most approximations can be simulated by simplifying Φ to $\xi(\Phi)$ and the use a normal reasoner:

$$\mathcal{T} \mid\!\sim_\xi \Phi \iff \mathcal{T} \mid\!- \xi(\Phi)$$

Given this correlation there is no practical need for an approximated inference engine for the basic inference. In order to approximate the DL reasoning it is enough in most cases to only approximate the complex inference and to approximate the basic inference by simplifying the checked expression. For the satisfiability test a normal DL reasoner can still be used. By this approach many different approximation methods can be investigated and evaluated without the expensive cost for the development of a full (but approximated) DL reasoner. Furthermore the optimizations and the improvements of current DL reasoner are still accessible.

7.3.1 Approximating Classification

As an example for an Approximate Deduction we explain the implementation of the approximate classification described in D2.1.2 "Methods for Approximate Reasoning". The approximation method proposed by Cadoli and Schaerf [SC95] was applied to the classification of a number of concepts and evaluated for realistic ontologies. Cadoli and Schaerf [SC95] propose a method which starts with a maximally simplified expression and checks for satisfiability. If this test fails the next more complex expression is checked. This proceeds until the first test succeeds or until the original expression is reached.

The normal classification procedure does not have to be changed (Algorithm 1, see D2.1.2 "Methods for Approximate Reasoning"). For classifying a concept expression Q

Algorithm 1 classification

```
Require: A classified concept hierarchy with root $\top$
Require: A query concept $Q$
  Visited := $\varnothing$
  Result := $\varnothing$
  Goals := $\{\top\}$
  while Goals $\neq \varnothing$ do
    C$\in$Goals where $\{$direct parents of C$\} \subseteq$ Visited
    Goals := Goals $\setminus \{$C$\}$
    Visited := Visited $\cup \{$C$\}$
    if unsatisfiable($Q \sqcap \neg C$) then
      Goals := Goals $\cup \{$direct children of C$\}$
      Result := (Result $\cup \{$C$\}) \setminus \{$all ancestors of C$\}$
    end if
  end while
  Eliminate equal concepts in Result
  if $|$Result$|$ = 1 $\wedge$ unsatisfiable($C \sqcap \neg Q$) then
    Equal := 'yes'
  else
    Equal := 'no'
  end if
  return Equal, Result
```

into the concept hierarchy a number of subsumption tests are generated. The most specific concepts w.r.t. the subsumption hierarchy which passed the subsumption test are collected for the results. During classification a sequence of satisfiability test is generated.

For the approximated classification the satisfiability test has to be replaced. Two different approximations, the $C^\top$- and $C^\perp$-satisfiability, are implemented by the Algorithms 2 and 3. The approximations are easily constructed in linear time. In contrast to the normal classification each satisfiability test is replaced by a series of satisfiability test. When the approximation at a certain level I does not lead to a conclusion the level I is increased by one and the next satisfiability test of the series is generated. Please note that the result of the approximated satisfiability tests for both kinds of approximations must be different.

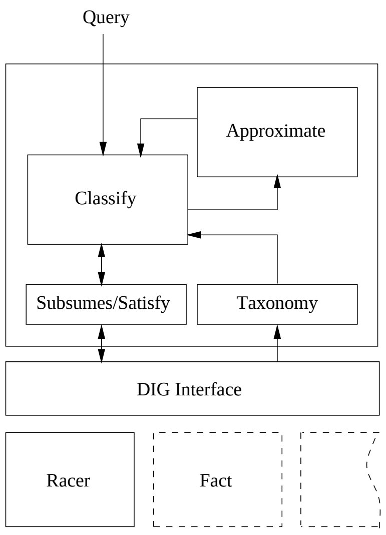


Figure 7.1: Architecture of experimental setup.

Algorithm 2 $C^{\top}$-satisfiability

```
Require: A complex concept expression $C$
Require: A Query $Q$
  I := 0
  repeat
    Current := $(Q \sqcap \neg C)^{\top}_{I}$
    Result := unsatisfiable(Current)
    if Result = ‘true’ then
      break
    end if
    I := I+1
  until Current = $Q \sqcap \neg C$
  return Result
```

Algorithm 3 $C^{\perp}$-satisfiability

```
Require: A complex concept expression $C$
Require: A Query $Q$
  I := 0
  repeat
    Current := $(Q \sqcap \neg C)^{\perp}_{I}$
    Result := unsatisfiable(Current)
    if Result = ‘false’ then
      break
    end if
    I := I+1
  until Current = $Q \sqcap \neg C$
  return Result
```

The replacement of a particular satisfiability test with a series of satisfiability tests may indicate that reasoning performance can not be improved. But each satisfiability test from the series is performed on a simplified expression which implies a cheaper satisfiability test. Cadoli and Schaerf [SC95] hope that a series of cheap tests can beat one expensive test — the replaced original test. Unfortunately first experiments reported in D2.1.2 “Methods for Approximate Reasoning” do not support their thesis.

Each DL reasoner (e.g., Fact [Hor98a], Fact++ (see chapter 3), or Racer [HM01a]) implements the classification functionality internally. In order to approximate classification independently from the implementation of a particular DL reasoner, the classification method must be (re)implemented. The classification procedure must be built on top of an arbitrary DL reasoner according to Algorithm 1. The satisfiability tests are propagated to the DL reasoner through the DIG interface [BMC03b] as depicted in Figure 7.1. The approximated classifier can also act as a separate server; if only it implements a DIG interface and acts as HTTP-server waiting for queries.

7.4 Conclusion

We gave a brief overview over architectures to approximations for Semantic Web reasoning. Language Weakening and Knowledge Compilation need a translator which transform the ontology and query into a new representation. Approximate Deduction can concentrate on approximate complex inferences which are reduced to satisfiability tests. But in most cases a mixture of these architectures will occur. As shown in the case of Knowledge Compilation showed (see D2.1.2 “Methods for Approximate Reasoning”), where an OWL DL ontology is compiled into a disjunctive logic program, the inference itself can also be approximated. In this example the disjunctions in head of the clauses in the logic program are ignored. It is also a further example for a strong correlation between approximate inference and simplified expressions for normal inferences.

However the approximate deduction assumes one query which will be approximated. But in many cases the application will generate many very similar queries. Instead of sending each query one by one the application should send all queries together and let the reasoner decide in which order and to which degree the queries will be evaluated and answered. Surely such a strategy provides more potential for approximation.

Chapter 8

Querying Answering with Instance Store

8.1 Introduction

In this chapter, we present the *instance Store* (*iS*) system, which provides efficient retrieval using a hybrid database/reasoner architecture: a relational database is used to persist instances, while a TBox reasoner is used to infer ontological information about the classes they belong to; moreover, part of this ontological information is also persisted in the database.

The *iS* only supports a very limited form of reasoning about individuals, i.e., answering instance retrieval queries w.r.t. an ontology and a set of axioms asserting class-instance relationships. With this restriction it is not possible to assert, e.g., that John is the brother of Peter (where John and Peter are individuals), but it *is* possible to assert, e.g., that Peter has at least three brothers, all of whom are either Doctors or Dentists. This kind of reasoning turns out to be useful in a wide range of applications, in particular those where domain models are used to structure and investigate large data sets. In the Gene Ontology (GO) application described below (in the section on Empirical Evaluation), for example, a complex model of gene structure and function is used to annotate and query gene product data; the gene product *1433 CANAL* is, for example, described as an instance of the class of gene products that **take part in** *intracellular signalling cascade*, are **part of** *chloroplast*, and have the **function** of *protein domain specific binding activity* (where properties are shown in bold and classes in italics).

It is clear that, from a theoretical point of view, this functionality could be reduced to pure TBox reasoning.[1] The *iS* is, however, able to deal with *much* larger numbers of individuals than would be possible using a standard Description Logic reasoner.

[1]Answering such queries is not trivial, however, as the query class can be a complex description that does not occur in the ontology.

In order to evaluate the *iS* design, and in particular its ability to provide scalable performance for instance retrieval queries, we have performed a number of experiments using the *iS* to search over a large gene ontology and its associated very large number of individuals—instances of concept descriptions formed using terms from the ontology.

8.2 Instance Store

An ABox $\mathcal{A}$ is role-free if it contains only axioms of the form $x : C$, where x is an individual and C is a (possibly complex) concept. We can assume, without loss of generality, that there is exactly one such axiom for each individual as $x : C \sqcup \neg C$ holds in all interpretations, and two axioms $x : C$ and $x : D$ are equivalent to a single axiom $x : (C \sqcap D)$. It is well known that, for a role-free ABox, instantiation can be reduced to TBox subsumption [Hol96, Tes97]; i.e., if $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, and $\mathcal{A}$ is role-free, then $\mathbf{K} \models x : D$ iff $x : C \in \mathcal{A}$ and $\mathcal{T} \models C \sqsubseteq D$. Similarly, if $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ and $\mathcal{A}$ is a role-free ABox, then the instances of a concept D could be retrieved simply by testing for each individual x in $\mathcal{A}$ if $\mathbf{K} \models x : D$. However, this would clearly be very inefficient if $\mathcal{A}$ contained a large number of individuals.

An alternative approach is to add a new axiom $C_x \sqsubseteq D$ to $\mathcal{T}$ for each axiom $x : D$ in $\mathcal{A}$, where C_x is a new atomic concept; we will call such concepts *pseudo-individuals*. Classifying the resulting TBox is equivalent to performing a complete realisation of the ABox: the most specific atomic concepts that an individual x is an instance of are the most specific atomic concepts that subsume C_x. Moreover, the instances of a concept D can be retrieved by computing the set of pseudo-individuals that are subsumed by D.

The problem with this latter approach is that the number of pseudo-individuals added to the TBox is equal to the number of individuals in the ABox, and if this number is very large, then TBox reasoning may become inefficient or even break down completely (e.g., due to resource limits).

We now introduce some new notation used, for convenience, in this paper. For $\mathcal{T}$ a TBox, $\mathbf{A}$ the set of atomic concepts (concept names) occurring in $\mathcal{T}$, and C a (possibly complex) concept, we define:

- $C\downarrow_{\mathcal{T}} = \{a \mid a \in \mathbf{A} \text{ and } a \sqsubseteq C\}$, i.e., $C\downarrow_{\mathcal{T}}$ is the set whose members are the atomic concepts in $\mathcal{T}$ that are subsumed by C (note that this could include concepts that are equivalent to C).

- $\lceil C \rceil_{\mathcal{T}} = \{a \mid a \in \mathbf{A}, C \sqsubseteq a \text{ and there is no } a' \in \mathbf{A} \text{ with } C \sqsubseteq a' \text{ and } a' \sqsubseteq a\}$, i.e., $\lceil C \rceil_{\mathcal{T}}$ is the set whose members are the most specific atomic concepts in $\mathcal{T}$ that subsume C (note that if C is itself an atomic concept in $\mathcal{T}$, then $\lceil C \rceil_{\mathcal{T}} = \{C\}$).

The basic idea behind the *iS* is to overcome this problem by using a DL reasoner to classify the TBox and a database to store the ABox. Our starting point is the 'semantic indexing' of [Sch94], taking the atomic concepts in the ontology as indexing concepts, so we use the database also to store a complete realisation of the ABox, i.e., for each individual x, the concepts that x realises (the most specific atomic concepts that x instantiates). The realisation of each individual is computed using the DL (TBox) reasoner when an axiom of the form $x : C$ is added to the *iS* ABox.

A retrieval query Q to the *iS* (i.e., computing the set of individuals that instantiate a concept Q) can be answered using a combination of database queries and TBox reasoning. Given an *iS* containing a KB $\langle \mathcal{T}, \mathcal{A} \rangle$ and a query concept Q, retrieval involves the computation of the following sets of individuals for which we introduce a special notation:

- I_1 denotes the set of individuals in $\mathcal{A}$ that realise *some* concept in $Q\downarrow_{\mathcal{T}}$;
- I_2 denotes the set of individuals in $\mathcal{A}$ that realise *every* concept in $\lceil Q \rceil_{\mathcal{T}}$.

The *iS* algorithm to retrieve the instances of Q can be then described as follows:

1. use the DL reasoner to compute $Q\downarrow_{\mathcal{T}}$;
2. use the database to retrieve the set of individuals I_1;
3. use the DL reasoner to check whether Q is equivalent to any atomic concept in $\mathcal{T}$; if so then return I_1 and *exit*;
4. use the DL reasoner to compute $\lceil Q \rceil_{\mathcal{T}}$;
5. use the database to retrieve the set of individuals I_2;
6. use the DL reasoner to check if Q subsumes (hence is equivalent to) the conjunction of the concepts in $\lceil Q \rceil_{\mathcal{T}}$; if so then return $I_1 \cup I_2$ and *exit*;
7. use the DL reasoner and the database to compute I_3, the set of individuals $x \in I_2$ such that $x : C$ is an axiom in $\mathcal{A}$ and C is subsumed by Q;
8. return $I_1 \cup I_3$ and *exit*.

Proposition. The above procedure is sound and complete for retrieval, i.e., given a concept Q, it returns all and only individuals in $\mathcal{A}$ that are instances of Q.

The above is easily proved using the fact that we assume, without loss of generality, that for each individual x there is exactly one axiom of the form $x : C$ in $\mathcal{A}$.

8.2.1 An Optimised Instance Store

In practice, several refinements to the above procedure are used to improve the performance of the *iS*. In the first place, as it is potentially costly, we should try to minimise the DL reasoning required in order to compute realisations (when instance axioms are added to the ABox) and to check if individuals in I_2 are instances of the query concept (when answering a query).

One way to (possibly) reduce the need for DL reasoning is to avoid repeating computations for "equivalent" individuals, e.g., individuals x_1, x_2 where $x_1 : C_1$ and $x_2 : C_2$ are ABox axioms, and C_1 is equivalent to C_2. Since checking for semantic equivalence between two concepts would require DL reasoning (which we are trying to avoid), the optimised *iS* only checks for syntactic equality using a database lookup. (The chances of detecting equivalence via syntactic checks could be increased by transforming concepts into a syntactic normal form, as is done by optimised DL reasoners [Hor03], but this additional refinement has not yet been implemented in the *iS*.) Individuals are grouped into equivalence sets, where each individual in the set is asserted to be an instance of a syntactically identical concept, and only one representative of the set is added to the *iS* ABox as an instance of the relevant concept. When answering queries, each individual in the answer is replaced by all the individuals in its equivalence set.

Similarly, we can avoid repeated computations of sub and super-concepts for the same concept (e.g., when repeating a query) by caching the results of such computations in the database.

Finally, the number and complexity of database queries also has a significant impact on the performance of the *iS*. In particular, the computation of I_1 can be costly as $Q \downarrow_{\mathcal{T}}$ may be very large. One way to reduce this complexity is to store not only the most specific concepts instantiated by each individual, but to store *every* concept instantiated by each individual. As most concept hierarchies are relatively shallow, this does not increase the storage requirement too much, and it greatly simplifies the computation of I_1: it is only necessary to compute the (normally) much smaller set of most general concepts subsumed by Q and to query the database for individuals that instantiate some member of this set. On the other hand, the computation of I_2 is slightly more complicated as I_1 must be subtracted from the set of individuals that instantiate every concept in $\lceil Q \rceil_{\mathcal{T}}$. Empirically, however, the saving when computing I_1 seems to far outweigh the extra cost of computing I_2.

8.3 Implementation

We have implemented the *iS* using a component based architecture that is able to exploit existing DL reasoners and databases. The core component is a Java application talking to a DL reasoner via the DIG interface [Bec03] and to a relational database via JDBC. We have tested it with FaCT [Hor98b], FaCT++ [Tsa05] and RACER [HM01d] reasoners and MySQL, Hypersonic, and Oracle databases.

```
initialise(Reasoner, Database, TBox)
addAssertion(Individual, Concept)
retract(Individual)
retrieve(Concept): Set⟨Individual⟩
```

Figure 8.1: Basic functionality of *iS*

The basic functionality of the *iS* is illustrated in Figure 8.1. The four basic operations are `initialise`, which loads a TBox into the DL reasoner, classifies the TBox and establishes a connection to the database; `addAssertion`, which adds an axiom $i : D$ to the *iS*; `retract`, which removes any axiom of the form $i : C$ (for some concept C) from the *iS*; and `retrieve`, which returns the set of individuals that instantiate a query concept Q. As the *iS* ABox can only contain one axiom for each individual, asserting $i : D$ when $i : C$ is already in the ABox is equivalent to first removing $i : C$ (using `retract`) and then asserting $i : (C \sqcap D)$.

In the current implementation, we make the simplifying assumption that the TBox itself does not change. Extending the implementation to deal with monotonic extensions of the TBox would be relatively straightforward, but deleting information from the TBox might require (in the worst case) all realisations to be recomputed.

8.4 Empirical Evaluation

In order to test the performance of the *iS*, we chose a real world problem with more that half a million instances. The Gene Ontology (*GO*) consortium publishes every month a database [Gen03] of gene products referring to terms in a large (25,180 concepts) ontology. The structural simplicity of the ontology (little more than a taxonomy of classes) means that its transitive closure can be precomputed and stored in the database so that, when a client searches for the gene products whose descriptions are subsumed by a set of terms, the answer can be returned without using a reasoner. Together with other functionality provided by the database, this provides biologists with a service which is highly valued and widely used.

We built a GO *iS* by mining (the SWISS-PROT fragment of) the Gene Ontology database and extracting 653,762 gene product descriptions which we loaded into the *iS* using the addAssertion method. In our mining we exploited the fact that gene terms form three more or less separate taxonomies of ‘processes’, ‘components’ and ‘functions’. We therefore added three corresponding new properties (i.e., DL roles) to the gene ontology and described gene products using them. For instance, we asserted that *1433 CANAL* is an instance of the class of gene products that **take part in** *intracellular signalling cascade*, are **part of** *chloroplast*, and have the **function** of *protein domain specific binding activity*. (We denote roles in bold and concepts in italics.)

This does not take into account annotations and other information present in the GO database, but our aim was simply to test a large set of realistic and interesting data. Extensions in the structure of the ontology (as envisaged in GONG [WSGA03]) would allow more complex assertions to be made and more complex queries to be asked.

As well as providing an enhanced query answering mechanism for GO annotated gene products, bioinformatics applications of the *iS* include its use to guide gene annotation [BTMS04] and, more recently, to investigate the structure of data mined from the InterPro database of protein families.

Another example *iS* application is in the *MONET* project [CDT04], where it is used in a web-services broker. MONET envisages mathematical web-services being registered with a broker, along with ontology based descriptions of their capabilities, and discovered by clients using service requests consisting of similar descriptions of the required functionality. The *iS* has been used to perform the matching of service requests against capability descriptions in a proof-of-concept prototype of the broker. A typical service description specifies the 'GAMS' classification of the service, the problem it solves, input and output formats, the directives it accepts, the software used to implement it, and the algorithm it implements. All this involves several classes and roles in nested conjunctions from an ontology containing thousands of classes interconnected by means of tens of roles. The structural richness of the ontology means that services can then be matched using, e.g., a bibliographic reference to their implemented algorithm. The MONET *iS* contains too few instances for its performance to be an issue, but it does illustrate the expressive possibilities of the *iS* approach.

GO Tests

We have tested the retrieval performance of the GO *iS* using a set of queries; their descriptions are similar in structure to the description of the assertion for the above gene product *1433 CANAL*, i.e. a conjunction of processes, components and functions (each conjunct possibly empty). The query set was formulated with the help of domain experts and consists of twelve queries that might be posed by a biologist. They are designed to test the effect on query answering performance of factors such as the number of individuals in the answer, whether the query concept is equivalent to an atomic concept (if so, then the answer can be returned without computing I_2 or I_3), and the number of candidate individuals in I_2 for which DL reasoning is required in order to determine if they form part of the answer. The characteristics of the various queries with respect to these factors is shown in Table 8.1.

The tests were performed using Linux, Intel Pentium 4 CPU 2.40GHz, 256MB RAM, MySQL 4.1.10a, and FaCT++ 0.99.3.

In these tests, we explore the performance of the *iS* using the GO TBox and differently sized and randomly selected subsets of the GO ABox. The *iS* was first initialised with the GO TBox (it took FaCT++ approximately 85 CPU seconds to classify the TBox), then,

Table 8.1: Query characteristics

Query	Equivalent to Atomic Concept	No. of Instances in Answer	No. of "candidates" in I_2
Q1	Yes	6,527	0
Q2	No	4	19
Q3	No	13	0
Q4	Yes	96,105	0
Q5	Yes	27	0
Q6	No	13,449	0
Q7	No	11,820	0
Q8	No	12	604
Q9	No	19	0
Q10	Yes	4,563	0
Q11	Yes	1	0
Q12	No	16	7,867

for each ABox, we measured the time (in CPU seconds) taken to load the ABox into the *iS* and the time taken to answer each of the queries.

Table 8.2: *iS* load and realise times (CPU seconds)

Number of Individuals	Distinct Descriptions	Load & Realise
100	88	5
500	330	18
1,000	591	34
5,000	2,014	207
10,000	3,299	507
50,000	9,853	1,947
100,000	15,181	3,555
200,000	23,564	7,071
400,000	35,964	14,281
653,762	48,584	23,790

The time taken by the *iS* to load and realise the various ABoxes are shown in Table 8.2. Note that it increases more slowly than the size of the ABox: for ABox size 100, the *iS* takes about 50ms to add each individual axiom; by the time the ABox size has reached 653,762 this has fallen to approximately 35ms per axiom. In view of the equivalent individuals optimisation employed by the *iS*, however, it may be more relevant to consider the time taken per distinct description: this increases from about 60ms per description for the size 100 ABox (which contains 88 distinct descriptions) to approximately 0.5s per description for the size 653,762 ABox (which contains 48,584 distinct descriptions).

Note the load/realise operation only needs to be performed once—an added advantage of the *iS* is that the database provides for persistence of the realised ABox. Depending on the nature of the application, it may also be more normal for instance data to be added to the *iS* over time rather than all at once as in our experiment.

Tables 8.3 and 8.4 give the results for the *iS* when answering each of the twelve queries described in Table 8.1. In addition to the time taken (in CPU seconds) to answer the queries, the number of candidate individuals in I_2 is also given (where this is non-zero) as this is one of the major factors in determining the “hardness” of the query: for each individual in I_2, the *iS* must use the DL reasoner to determine if the individual instantiates the query concept. The time taken to answer these queries is also plotted against the size of the ABox in Figure 8.2; note the logarithmic scales on both axes.

Table 8.3: *iS* query times (CPU seconds) and cardinality of I_2 for queries Q1–Q7

Indiv'ls	Q1	Q2		Q3	Q4	Q5	Q6	Q7
	IS	IS	$\|I_2\|$	IS	IS	IS	IS	IS
100	0.2	1.0	1	0.1	0.1	0.1	0.1	0.1
500	0.5	1.1	2	0.1	0.3	0.1	0.1	0.1
1,000	1.1	1.3	2	0.1	0.3	0.2	0.1	0.2
5,000	1.7	1.5	2	0.1	0.3	0.4	0.5	0.3
10,000	0.8	1.3	2	0.1	0.6	0.1	0.7	0.2
50,000	1.7	1.6	3	0.4	1.8	0.2	1.8	0.4
100,000	2.5	1.8	4	0.5	3.0	0.2	2.9	0.6
200,000	4.1	2.5	4	1.0	6.0	0.3	5.9	1.6
400,000	6.3	3.3	10	1.6	12.0	0.3	10.0	2.0
653,762	6.9	5.3	19	2.4	17.8	0.6	16.5	2.9

Table 8.4: *iS* query times (CPU seconds) and cardinality of I_2 for queries Q8–Q12

Indiv'ls	Q8		Q9	Q10	Q11	Q12	
	IS	$\|I_2\|$	IS	IS	IS	IS	$\|I_2\|$
100	0.1	1	0.1	0.1	0.1	0.3	14
500	0.1	1	0.1	0.1	0.1	0.9	45
1,000	0.1	3	0.1	0.1	0.1	1.5	82
5,000	1.1	15	0.3	0.1	0.1	3.8	269
10,000	0.9	21	0.3	0.1	0.1	5.8	460
50,000	3.4	90	1.3	0.1	0.1	18.6	1,459
100,000	5.9	140	2.6	0.6	0.1	28.0	2,313
200,000	12.9	255	5.8	1.3	0.1	41.6	3,673
400,000	21.6	435	10.3	2.0	0.1	57.9	5,780
653,762	36.0	604	12.8	3.8	0.1	80.0	7,867

As can be seen, the time taken to answer queries becomes quite large when the number of individuals in I_2 is large. In these cases, the time taken to check if these individuals instantiate the query concept (roughly 0.01s per individual) dominates other factors. The number of “distinct” individuals in the answer also has a significant impact on query answering performance: when there are many such individuals, the database query required in order to compute the complete answer set (i.e., retrieving the union of the equivalence sets of these individuals) can be quite time consuming. In the case of Q6 with the largest ABox, for example, the relevant database query takes 14.7s (out of a total of 16.5s).

When the query concept is determined to be semantically equivalent to an atomic concept in the TBox, as is the case with Q1, Q4, Q5, Q10 and Q11, then no further DL reasoning is required. In these cases, the time taken to answer the query changes much more slowly with ABox size, and is mainly determined by the answer size. With Q4, for example, the time taken to answer the query rises to over 17s with the largest ABox, when the answer contains 96,105 individuals.

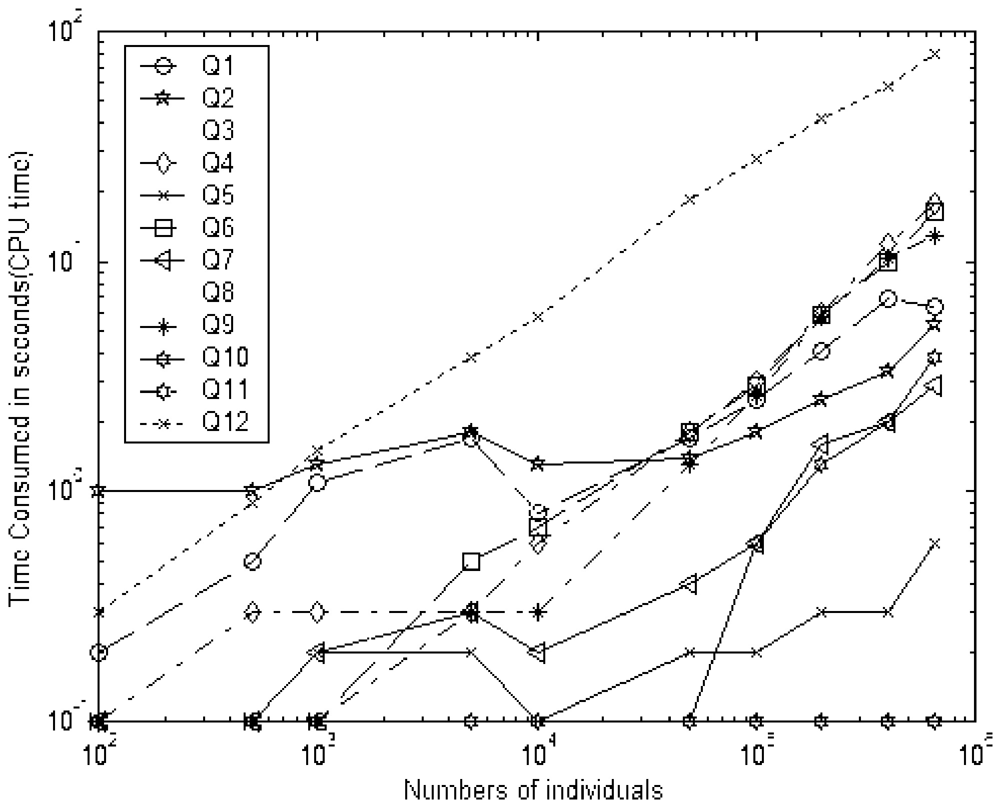


Figure 8.2: *iS* query times -v- ABox size

8.5 Related and Future Work

There is a long tradition of coupling databases to knowledge representation systems in order to perform reasoning, most notably the work in [BB93]. However, in our architecture we do not use the standard approach of associating a table (or view) with each class and property. Instead, we have a fixed and relatively simple schema that is independent of the structure of the ontology and of the instance data. The *iS* is, therefore, agnostic about the provenance and structure of data: it uses a new, dedicated database for each ontology, but the schema is always the same.

Another example is the Parka system [ASH95]. Parka is not limited to role-free ABoxes and can deal with very large ABoxes. However, Parka also supports a much less expressive description language, and is not based on standard DL semantics, so it is not really comparable to the *iS*.

The architectural choices made in the implementation of the *iS* ensure that we use appropriate technologies for appropriate tasks. It is clear that at some point the reasoner must be used in order to retrieve individuals, but in our approach it is only used when necessary. Databases are well suited to handling large volumes of data and are optimised for the performance of operations such as joins and intersections.

The functionality of the *iS* is limited, but is sufficient to support several interesting applications, and allows us to deal with volumes of instance data that cannot, to the best of our knowledge, be handled by any other system that would guarantee sound and complete query answering.

In the present *iS*, roles are allowed to appear at the class level, as in the GO role **take part in**, but no role assertion between instances is allowed, i.e., we cannot assert that instance x is related via role **r** to instance y. We are currently working on an extension of the *iS* that uses the *precompletion* technique [Hol96] to overcome this limitation (although at the cost of some restrictions on the structure of the ontology).

Chapter 9

Conclusion

In this report, we have investigated some aspects of implementation and optimisation techniques of Semantic Web query systems.

Firstly, we have provided a brief survey of some Semantic Web query engines, comparing some well known query engines, including Jena, Sesame, RdfSuite, Triple and Racer. This survey shows that some engines fail to pass our tests even within RDFS. *Within the systems that we tested*, Racer (from the DL community) is the only query system that passes all our tests on RDFS and OWL ontologies.

As system architectures play an important role in Semantic Web reasoning and querying systems, we have covered this topic with two chapters. In Chapter 3, we have presented the ToDo list architecture, which is implemented in the FaCT++ system. This architecture allows different priority ordering of rule-ordering heuristics, which can be very effective on performance improvements. In Chatper 4, we have provided a bigger picture, where datatype reasoners are introduced into DL systems. This architecture, implemented in the FaCT-DG DL reasoner, is very flexible in that different datatype checkers can be easily plugged in or taken out of the system, depending on the needs of applications. Equally importantly, to the best of our knowledge, FaCT-DG is the first DL reasoner that supports customised datatypes and datatype predicates.

Thirdly, we have also presented algorithms of reasoning with the f-$\mathcal{SI}$ DL. Reasoning with fuzzy DLs is a very popular research topic these days. To the best of our knowledge, we are the first to provide reasoning support for fuzzy knowledge base satisfiability with the presence of fuzzy transitive role axioms.

Last but not least, we have also investigated some other aspects on implementations of Semantic Web query systems. Chapter 6 has described the principles of the design and development of intelligent query interface systems. Chapter 7 has proposed some ideas on three approaches to approximating DL reasoning, in particular the approximate deduction approach. Most importantly, in Chapter 8, we have presented how to combine Description Logic TBox reasoning and databases to facilitate efficient query answering of retrieval queries over extremely large numbers of individuals, which are not interconnected by role assertions.

As for future work, it would be interesting to investigate optimisations for role-enabled instance Store, as well as optimisation techniques for query answering over rule-extended knowledge bases.

Bibliography

[ABS03] J. Alejandro, Tseng Belle, and J. Smith. Modal keywords, ontologies and reasoning for video understanding. In *Proceedings of the International Conference on Image and Video Retrieval*, 2003.

[ACK$^+$01] Sofia Alexaki, Vassilis Christophides, Gregory Karvounarakis, Dimitris Plexousakis, and Karsten Tolle. The ics-forth rdfsuite: Managing voluminous rdf description bases. In *International Semantic Web COnference (SemWeb-01)*, 2001.

[ASH95] W. A. Andersen, K. Stoffel, and J. A. Hendler. Parka: Support for extremely large knowledge bases. In G. Ellis, R. A. Levinson, A. Fall, and V. Dahl, editors, *Knowledge Retrieval, Use and Storage for Efficiency: Proceedings of the First International KRUSE Symposium*, pages 122–133, 1995.

[Baa90] F. Baader. Augmenting Concept Languages by Transitive Closure of Roles: An Alternative to Terminological Cycles. Research Report RR-90-13, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH (DFKI), 1990. An abridged version appeaered in Proc. IJCAI-91,pp.446-451.

[BB93] Alexander Borgida and Ronald J. Brachman. Loading data into description reasoners. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 217–226, 1993.

[BCM$^+$03] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook*. CUP, 2003.

[Bec03] Sean Bechhofer. The DIG Description Logic Interface: DIG/1.1. URL `http://dl-web.man.ac.uk/dig/2003/02/interface.pdf`, Feb 2003.

[BF96] P. Bresciani and E. Franconi. Description logics for information access. In *Proceedings of the AI*IA 1996 Workshop on Access, Extraction and Integration of Knowledge*, Napoli, September 1996.

[BF02] Paolo Bresciani and Paolo Fontana. A knowledge-based query system for biological databases. In *Proceedings of FQAS 2002*, volume 2522 of *Lecture Notes in Computer Science*, pages 86–89. Springer Verlag, 2002.

[BFH$^+$94] Franz Baader, Enrico Franconi, Bernhard Hollunder, Bernhard Nebel, and Hans-Jürgen Profitlich. An empirical analysis of optimization techniques for terminological representation systems or: Making KRIS get a move on. *Applied Artificial Intelligence*, 4:109–132, 1994.

[BKvH02] J. Broekstra, A. Kampman, and F. van Harmelen. Sesame: A generic architecture for storing and querying rdf and rdf schema. In *International Semantic Web Conference (ISWC-02)*, 2002.

[Bl99] Tim Berners-lee. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web*. Harper, San Francisco, 1999.

[BL01] Tim Berners-Lee. Notation 3, 2001. `http://www.w3.org/DesignIssues/Notation3`.

[BLHL01] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, 2001.

[BMC03a] Sean Bechhofer, Ralf Mller, and Peter Crowther. The dig description logic interface. In *Proceedings of the 2003 International Workshop on Description Logics (DL2003)*, volume 81 of *CEUR Workshop Proceedings*, 2003.

[BMC03b] Sean Bechhofer, Ralf Möller, and Peter Crowther. The dig description logic interface. In *Proceedings of DL2003 International Workshop on Description Logics*, Rome, September 2003.

[BMNPS02] F. Baader, D. McGuinness, D. Nardi, and P.F. Patel-Schneider. *The Description Logic Handbook: Theory, implementation and applications*. Cambridge University Press, 2002.

[BMR99] Francesca Benzi, Dario Maio, and Stefano Rizzi. VISIONARY: a viewpoint-based visual language for querying relational databases. *J. Vis. Lang. Comput.*, 10(2):117–145, 1999.

[BNP00] Paolo Bresciani, Michele Nori, and Nicola Pedot. A knowledge based paradigm for querying databases. In *Database and Expert Systems Application*, volume 1873 of *Lecture Notes in Computer Science*, pages 794–804. Springer Verlag, 2000.

[Bra04] Sebastian Brandt. On Subsumption and Instance Problem in $\mathcal{ELH}$ w.r.t. General TBoxes. In *Proceedings of the 2004 International Workshop on Description Logics (DL2004)*, CEUR-WS, 2004.

[BSC00] A. B. Benitez, J. R. Smith, and S. Chang. MediaNet: a multimedia information network for knowledge representation. In *Proc. SPIE Vol. 4210, p. 1-12, Internet Multimedia Management Systems, John R. Smith; Chinh Le; Sethuraman Panchanathan; C.-C. J. Kuo; Eds.*, pages 1–12, October 2000.

[BSN$^+$99] Sean Bechhofer, Robert Stevens, Gary Ng, Alex Jacoby, and Carole A. Goble. Guiding the user: An ontology driven interface. In *UIDIS 1999*, pages 158–161, 1999.

[BTMS04] M. Bada, D. Turi, R. McEntire, and R. Stevens. Using Reasoning to Guide Annotation with Gene Ontology Terms in GOAT. *SIGMOD Record (special issue on data engineering for the life sciences)*, June 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.

[CCLB97] Tiziana Catarci, Maria Francesca Costabile, Stefano Levialdi, and Carlo Batini. Visual query systems for databases: A survey. *Journal of Visual Languages and Computing*, 8(2):215–260, 1997.

[CDD$^+$04] Jeremy J. Carroll, Ian Dickinson, Chris Dollin, Dave Reynolds, Andy Seaborne, and Kevin Wilkinson. Jena: implementing the semantic web recommendations. In *WWW (Alternate Track Papers & Posters)*, 2004.

[CDGL98] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. On the decidability of query containment under constraints. In *Proc. of PODS'98*, 1998.

[CDGL$^+$04] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Riccardo Rosati, and Guido Vetere. DL-Lite: Practical Reasoning for Rich DLs. In *Proc. of the 2004 Description Logic Workshop (DL 2004)*. CEUR Electronic Workshop Proceedings, `http://ceur-ws.org/Vol-104/`, 2004.

[CDR04] Jeremy Carroll and Jos De Roo. OWL web ontology language test cases. W3C Recommendation, 2004.

[CDT04] Olga Caprotti, Mike Dewar, and Daniele Turi. Mathematical service matching using Description Logic and OWL. In *Proceedings of 3rd International Conference on Mathematical Knowledge Management (MKM'04)*, volume 3119 of *LNCS*. Springer-Verlag, 2004.

[CGL$^+$98] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. Information integration: Conceptual modeling and reasoning support. In *Proc. of the 6th Int. Conf. on Cooperative Information Systems (CoopIS'98)*, pages 280–291, 1998.

[CGL00] D. Calvanese, G. De Giacomo, and M. Lenzerini. Answering queries using views over description logics knowledge bases. In *Proc. of the 16th Nat. Conf. on Artificial Intelligence (AAAI 2000)*, 2000.

[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. Available at `http://www.w3.org/TR/owl-ref/`.

[Fra00] Enrico Franconi. Knowledge representation meets digital libraries. In *Proc. of the 1st DELOS (Network of Excellence on Digital Libraries) workshop on "Information Seeking, Searching and Querying in Digital Libraries"*, 2000.

[Fre95] J. W. Freeman. *Improvements to Propositional Satisfiability Search Algorithms*. PhD thesis, University of Pennsylvania, 1995.

[Gen03] Gene Ontology Consortium. Gene Ontology Database. `http://www.godatabase.org/dev/database`, 2003.

[GGM$^+$02] Aldo Gangemi, Nicola Guarino, Claudio Masolo, Alessandro Oltramari, and Luc Schneider. Sweetening ontologies with DOLCE. In *Proc. of EKAW 2002*, 2002.

[GH04] Birte Glimm and Ian Horrocks. Query answering systems in the semantic web. In *CEUR workshop proceedings of KI-2004 Workshop on Applications of Description Logics (ADL 2004)*, September 24 2004.

[GHVD03] Benjamin N. Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. Description logic programs: Combining logic programs with description logic. In *Proc. of the Twelfth International World Wide Web Conference (WWW 2003)*, pages 48–57, 2003.

[GS96] Fausto Giunchiglia and Roberto Sebastiani. A SAT-based decision procedure for $\mathcal{ALC}$. In *Proc. of KR'96*, pages 304–314, 1996.

[Hal01] Terry A. Halpin. Augmenting UML with fact orientation. In *HICSS*, 2001.

[HKS02] Steffen Hölldobler, Tran Dinh Khang, and Hans-Peter Störr. A fuzzy description logic with hedges as concept modifiers. In *Proceedings InTech/VJFuzzy'2002*, pages 25–34, 2002.

[HM01a] V. Haarslev and R. Möller. Description of the racer system and its applications. 2001.

[HM01b] V. Haarslev and R. Möller. High performance reasoning with very large knowledge bases: A practical case study. In *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)*, 2001.

[HM01c] Volker Haarslev and Ralf Möller. Racer system description. In *Automated Reasoning: First International Joint Conference, IJCAR 2001*, volume 2083 of *Lecture Notes in Computer Science*. Springer-Verlag Heidelberg, 2001.

[HM01d] Volker Haarslev and Ralf Möller. RACER System Description. In *International Joint Conference of Automatic Reasoning (IJCAR2001)*, volume 2083, 2001.

[HM02] Volker Haarslev and Ralf Mller. Practical Reasoning in RACER with a Concrete Domain for Linear Inequations. In *Proceedings of the International Workshop on Description Logics (DL-2002), Toulouse, France.*, pages 91–98, April. 2002.

[HM03] V. Haarslev and R. Möller. Racer: A core inference engine for the semantic web. In *EON*, 2003.

[Hol96] Bernhard Hollunder. Consistency checking reduced to satisfiability of concepts in terminological systems. *Ann. of Mathematics and Artificial Intelligence*, 18(2–4):133–157, 1996.

[Hor97] Ian Horrocks. *Optimising Tableaux Decision Procedures for Description Logics*. PhD thesis, University of Manchester, 1997.

[Hor98a] I. Horrocks. The FaCT system. In H. de Swart, editor, *Automated Reasoning with Analytic Tableaux and Related Methods: International Conference Tableaux'98*, number 1397 in Lecture Notes in Artificial Intelligence, pages 307–312. Springer-Verlag, May 1998.

[Hor98b] I. Horrocks. Using an Expressive Description Logic: FaCT or Fiction? In *International Conference of Knowledge Representation (KR98)*, pages 636–647, 1998.

[Hor98c] I. Horrocks. Using an expressive description logic: FaCT or fiction? In *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR'98)*, pages 636–647, 1998.

[Hor99] I. Horrocks. FaCT and iFaCT. In *International Description Logics Workshop (DL99)*, pages 133–135, 1999.

[Hor03] I. Horrocks. Implementation and optimisation techniques. In Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation, and Applications*, pages 306–346. CUP, 2003.

[HPS98] I. Horrocks and P. F. Patel-Schneider. Comparing subsumption optimizations. In *International Description Logics Workshop (DL98)*, 1998.

[HPS99] I. Horrocks and P. F. Patel-Schneider. Optimising description logic subsumption. *Journal of Logic and Computation*, 9(3):267–293, 1999.

[HPS03a] Ian Horrocks and Peter F. Patel-Schneider. Reducing OWL entailment to description logic satisfiability. In *Proc. of the 2nd International Semantic Web Conference (ISWC)*, 2003.

[HPS03b] Ian Horrocks and Peter F. Patel-Schneider. Three theses of representation in the semantic web. In *Proc. of the Twelfth International World Wide Web Conference (WWW 2003)*, pages 39–47. ACM, 2003.

[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.

[HS99] I. Horrocks and U. Sattler. A description logic with transitive and inverse roles and role hierarchies. *Journal of Logic and Computation*, 9:385–410, 1999.

[HS02] Ian Horrocks and Ulrike Sattler. Optimised reasoning for $\mathcal{SHIQ}$. In *Proc. of the 15th Eur. Conf. on Artificial Intelligence (ECAI 2002)*, pages 277–281, July 2002.

[HST99] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Practical reasoning for expressive description logics. In *Proc. of LPAR'99*, pages 161–180, 1999.

[HST00] I. Horrocks, U. Sattler, and S. Tobies. Reasoning with Individuals for the Description Logic $\mathcal{SHIQ}$. In David MacAllester, editor, *CADE-2000*, number 1831 in LNAI, pages 482–496. Springer-Verlag, 2000.

[HSTT00] Ian Horrocks, Ulrike Sattler, Sergio Tessaris, and Stephan Tobies. How to decide query containment under constraints using a description logic. In *Logic for Programming and Automated Reasoning (LPAR 2000)*, volume 1955 of *Lecture Notes in Computer Science*, pages 326–343. Springer, 2000.

[HT02] Ian Horrocks and Sergio Tessaris. Querying the semantic web: a formal approach. In Ian Horrocks and James Hendler, editors, *Proc. of the 2002 International Semantic Web Conference (ISWC 2002)*, number 2342 in Lecture Notes in Computer Science. Springer-Verlag, 2002.

[JW90] R. Jeroslow and J. Wang. Solving propositional satisfiability problems. *Ann. of Mathematics and Artificial Intelligence*, 1:167–187, 1990.

[LH03] Lei Li and Ian Horrocks. A Software Framework For Matchmaking Based on Semantic Web Technology. In *Proc. of the Twelfth International World Wide Web Conference (WWW 2003)*, pages 331–339. ACM, 2003.

[Mas99] Fabio Massacci. TANCS non classical system comparison. In *Proc. of TABLEAUX'99*, 1999.

[MGP98] Norman Murray, Carole Goble, and Norman Paton. A framework for describing visual interfaces to databases. *J. Vis. Lang. Comput.*, 9(4):429–456, 1998.

[MH03] Ralf Moller and Volker Haarslev. Description Logic Systems. In Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation, and Applications*, pages 282–305. Cambridge University Press, 2003.

[MM04] Frank Manola and Erik Miller. Rdf primer, 2004. `http://www.w3.org/TR/rdf-primer/`.

[MvH04] Deborah L. McGuinness and Frank van Harmelen. OWL Web Ontology Language Overview . Technical report, W3C Recommendation, Feb 2004.

[ORM03] `http://www.orm.net`, 2003.

[Pan04] Jeff Z. Pan. *Description Logics: Reasoning Support for the Semantic Web*. PhD thesis, School of Computer Science, The University of Manchester, 2004.

[PH05] Jeff Z. Pan and Ian Horrocks. OWL-Eu: Adding Customised Datatypes into OWL. In *Proc. of Second European Semantic Web Conference (ESWC 2005)*, 2005.

[PS99] Peter F. Patel-Schneider. DLP. In *Description Logics*, 1999.

[RDF01] http://lists.w3.org/archives/public/www-rdf-logic/. W3C Mailing List, starts from 2001.

[RRS$^+$01] J. E. Rogers, A. Roberts, W. D. Solomon, E. van der Haring, C. J. Wroe, P. E. Zanstra, and A. L. Rector. GALEN ten years on: Tasks and supporting tools. In *Proc. of MEDINFO2001*, pages 256–260, 2001.

[SC95] M Schaerf and M Cadoli. Tractable reasoning via approximation. *Artificial Intelligence*, 74:249–310, 1995.

[Sch94] A. Schmiedel. Semantic indexing based on description logics. In F. Baader, M. Buchheit, M.A. Jeusfeld, and W. Nutt, editors, *Reasoning about structured objects: knowledge representation meets databases. Proceedings of the KI'94 Workshop KRDB'94*. CEUR, September 1994.

[SD02] M. Sintek and S. Decker. Triple–a query, inference, and transformation language for the semantic web. In *International Semantic Web Conference (ISWC-02)*, 2002.

[Sem04] http://lists.w3.org/archives/public/public-swbp-wg/. W3C Mailing List, starts from 2004.

[SSS91] M. Schmidt-Schauß and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48:1–26, 1991.

[SST$^+$05] Giorgos Stoilos, Giorgos Stamou, Vassilis Tzouvaras, Jeff Pan, and Ian Horrocks. A fuzzy description logic for multimedia knowledge representation. In *Proceedings of the European Workshop on Multimedia and the Semantic Web*, 2005.

[ST04] D. Sánchez and G.B. Tettamanzi. Generalizing quantification in fuzzy description logic. In *Proceedings 8th Fuzzy Days in Dortmund*, 2004.

[Str01] U. Straccia. Reasoning within fuzzy description logics. *Journal of Artificial Intelligence*, 14:137–166, 2001.

[Str04] Umberto Straccia. Transforming fuzzy description logics into classical description logics. In *Proceedings of the 9th European Conference on Logics in Artificial Intelligence (JELIA-04)*, number 3229 in Lecture Notes in Computer Science, pages 385–399, Lisbon, Portugal, 2004. Springer Verlag.

[Tes97] Sergio Tessaris. *Questions and answers: reasoning and querying in Description Logic.* PhD thesis, The University of Manchester, 1997. URL `http://www.cs.man.ac.uk/~tessaris/papers/phd-thesis.ps.gz` .

[TH00] Anni-Yasmin Turhan and Volker Haarslev. Adapting Optimization Techniques to Description Logics with Concrete Domains. In *Proceedings of the International Workshop in Description Logics 2000 (DL2000), Aachen, Germany*, pages 247–256, 2000.

[TM98] C. Tresp and R. Molitor. A description logic for vague knowledge. In *In proc of the 13th European Conf. on Artificial Intelligence (ECAI-98)*, 1998.

[Tsa05] D. Tsarkov. FaCT++. `http://owl.man.ac.uk/factplusplus`, 2005.

[Ull97] J. D. Ullman. Information integration using logical views. In *Proc. of the 6th Int. Conf on Database Theory (ICDT'97)*, pages 19–40, 1997.

[VH01] Anni-Yasmin Turhan Volker Haarslev, Ralf Mller. Exploiting Pseudo Models for TBox and ABox Reasoning in Expressive Description Logics. In *Proceedings of International Joint Conference on Automated Reasoning, IJCAR'2001, R. Gor, A. Leitsch, T. Nipkow (Eds.),Siena, Italy, Springer-Verlag, Berlin.*, pages 61–75, Jun. 2001.

[WSGA03] Chris J. Wroe, Robert D. Stevens, Carole A. Goble, and Michael Ashburner. A methodology to migrate the gene ontology to a description logic environment using daml+oil. In *Proceedings of the 8th Pacific Symposium on Biocomputing (PSB)*, Hawaii, January 2003.

[Zim87] H.J. Zimmermann. *Fuzzy Sets, Decision Making, and Expert Systems*. Kluwer, Boston, 1987.