knowledgeweb
realizing the semantic web

D2.1.2.2.v1 Report on realizing practical approximate and distributed reasoning for ontologies

Stefania Ghita-Costache (Research Center L3S)
Peter Dolog (Research Center L3S)
Pascal Hitzler (Universität Karlsruhe)
Luciano Serafini (Centro per la ricerca scientifica e tecnologica)
Wolf Siberski (Research Center L3S)
Heiner Stuckenschmidt (Vrije Universiteit Amsterdam)
Andrei Tamilin (Centro per la ricerca scientifica e tecnologica and University of Trento)
Denny Vrandecic (Universität Karlsruhe)
Holger Wache (Vrije Universiteit Amsterdam)

Abstract.
EU-IST Network of Excellence (NoE) IST-2004-507482 KWEB
Deliverable D2.1.2.2.v1 (WP2.1)
This document reports on practical applications of approximate reasoning, mainly based on A-Box techniques or cooperative query processing. It also shows examples and implementations of distributed reasoning systems for ontologies, as distributed instance retrieval or a mix of ontology querying and information retrieval methods.
Keyword list: approximation, reasoning, distribution, scalability

Document Identifier	KWEB/2005/D2.1.2.2.v1/1.0
Project	KWEB EU-IST-2004-507482
Version	v1/1.0
Date	Jan 30, 2006
State	final
Distribution	public

Knowledge Web Consortium

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Work package participants

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

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

Changes

Version	Date	Author	Changes
0.1	2005-10-12	Wolf Siberski	creation
0.2	2005-10-31	Pascal Hitzler	input chapter 3
0.3	2005-12-01	Stefania Ghita-Costache	integrated chapter 2, 4, 5
1.0	2005-12-09	Stefania Ghita-Costache	including the comments from Pascal Hitzler
1.1	2006-01-13	Stefania Ghita-Costache	including the comments from WPL Holger Wache
1.2	2006-01-29	Stefania Ghita-Costache	including the comments from QA Fausto Giunchiglia

Executive Summary

Solving the problem of scalability in ontologies is a must, since we are dealing with many novel applications that produce even larger ontologies and data sets. This deliverable continues the work of deliverable 2.1.2, in which we deal only with methods for approximate reasoning. This version not only focuses on practical applications for deploying such algorithms, but also handles the problem of distributed reasoning in ontologies and their further progresses.

In the beginning of this deliverable we concentrate upon various approximate reasoning methods proposed by our partners, as approximation techniques in instance retrieval, which make A-Box reasoning in Description Logics more scalable. We present SCREECH, a system which implements the same type of technique, but for OWL DL ontologies and a system applied in e-learning for making robust query processing over RDF heterogeneous data in order to provide personalized information access.

In the end we change the focus upon distributed reasoning in instance retrieval, as in large distributed environments, the process of introducing or removing new resources is always a problem, and therefore the scalability issue is ubiquitous. We introduce DRAGO and its applications in the Semantic Web, and an information retrieval system spread over a federated structure, able to give very good, easily scalable results.

Contents

Chapter 1

Introduction

by STEFANIA GHITA-COSTACHE & HOLGER WACHE

The main goal in the KnowledgeWeb work package 2.1 is to deliver viable solutions for the scalability problem in the Semantic Web, that is how to meet the growth requirements for computing solutions, without affecting their performance, emphasizing the fact that scalability needs robust and high-performance reasoning. One of the identified, more general solutions for this, rely on modularization, approximation and distribution of reasoning methods. In this deliverable we will focus on the last two solutions, concentrating more on practical implementations of them.

As already introduced in deliverable D2.1.2, approximation techniques can be distinguished in approaches which

- weaken the language,
- compile the knowledge,
- approximate the deduction, or
- combination of them.

In this deliverable we consider approximation approaches which compile the knowledge and/or approximate the deduction. Furthermore these investigated approximation approaches are restricted to the use cases of instance retrieval, i.e. A-Box reasoning in terms of DL reasoning, as it doesn't scale up well when the number of instances increases significantly. It is obvious that this use case we dominate the practical use of the Semantic Web in future.

The first contribution is in comparing the performance of two approximate reasoning methods, using Instance Store [56], developed to scale-up instance retrieval for such ontologies with a large number of instances, and Gene Ontology as benchmark data set. It continues the approximation effort already reported in D2.1.2. The second approach

also covers approximate A-Box reasoning for OWL DL ontologies, but using a different technique. The approximation method is based on the fact that reasoning with Horn logic is more efficient than reasoning with non-Horn knowledge bases. The idea is thus to use language compilation for transforming OWL DL ontologies into Disjunctive Horn clauses, or more precisely, into approximated Datalog rules. The compilation is mainly done by powerful KAON2 transformation algorithm; the additional part is performed by the SCREECH system which implements the approximation approach. SCREECH can report impressing performance results.

Instead of improving the performance we also focus on robust reasoning. In particular our approach investigates RDF query formulating, as basis for queries used on a large scale and as a necessity in order to provide high system scalability. By combining the two basic methods for cooperative query processing, query refinement and query relaxation, an implementation of a framework is proposed for information access, in order to provide robust, personalized access to heterogeneous RDF data, in the context of widely used querying over RDF data, but without the ability of users to formulate meaningful queries. The implementation mainly deals with conditional rewriting of rules for RDF query patterns and its application is discussed in the context of an e-learning system. The rewriting of queries is based on Event-Condition-Action [80] rules, in order to solve the problem of over-constraint queries.

This deliverable also concentrates upon providing solutions to the scalability problem in distributed environments, i.e. distributed ontologies, distributed resources of information. The first approach tries to reason about distributed ontologies: DRAGO is a distributed reasoning platform for distributed ontologies which are interrelated by semantic links. The theory is described behind performing distributed reasoning and simple distributed instance retrieval in Distributed Description Logics, the architecture and implementation of DRAGO and also some applications of the proposed model for the semantic web. DRAGO proves itself to perform well for the instance retrieval problem, making use of different semantic mappings and reasoning on top of them.

The second approach broaden the view on distributed reasoning. It continues along the distributed resources paradigm, especially into a peer-to-peer network, by combining ontology-based querying and classic information retrieval methods in such an environment. The main idea is creating a local indexing scheme that actually indexes resources, diminishes the message overhead in the network and makes use of the collection-wide information efficiently. Additionally, this method allows easy dissemination of newly introduced information resources among the participating peers. Also, the introduction of the idea of a separate category index allows the reduction of large portions of the network without affecting quality, and therefore enhancing even more scalability.

Chapter 2

Scalable Instance Retrieval by Approximation

by HOLGER WACHE, PERRY GROOT & HEINER STUCKENSCHMIDT

2.1 Motivation

A central issue in the Semantic Web research community is the expressivity of its underlying language and the complexity of the reasoning services it supports. There is a direct correspondence between the current Semantic Web ontology language OWL and Description Logic (DL).[1] Research in DL has lead to sophisticated DL reasoners [48, 41, 43] that can be used to reason with OWL ontologies on the Semantic Web. Considering T-Box reasoning, current state of the art techniques seem capable of dealing with real world ontologies [49, 42]. However, besides T-Box reasoning, an important application domain of ontologies is A-Box reasoning, i.e., reasoning and retrieving the individuals in an ontology. Experiments have shown that state of the art DL reasoners break down for A-Box reasoning when the number of instances becomes large [56]. Present work focuses at approximation techniques to make A-Box reasoning in DLs more scalable when retrieving instances from an ontology with a large number of instances. Approximation is a general technique that has been proven useful in many areas. The Semantic Web is no different, it is a typical application domain that can benefit from an approximate form of reasoning, which can deal with time pressure as well as other limited resources and is scalable to the vast amount of available information. These conditions hold in particular when instances need to be retrieved from an ontology.

Here we investigate optimization techniques that are based on approximate logical reasoning. The underlying idea of these techniques is to replace certain inference problems by simpler problems such that either the soundness or the completeness, but not

[1] More precisely two of the three species of OWL.

both, of the solutions is preserved. The solutions to the simpler problems are approximate solutions to the original problem.

The contribution of this work is in comparing the performance of two approximate reasoning methods proposed in the literature applied to the real world task of answering conjunctive queries over DL Knowledge Bases. For this, we used the Instance Store [56], a state of the art system developed to scale-up instance retrieval for ontologies with a large number of instances, and extended it with two approximation techniques. The Gene Ontology is used as benchmark data set to evaluate the performance of the approximation techniques.

The chapter is organized as follows. Section 2.2 defines the problem of instance retrieval in the context of Description Logics, which is restricted to conjunctive queries. Section 2.3 gives a brief overview of two approximation methods and describes how they can be applied to the problem of instance retrieval. Section 2.4 gives the results of experiments with the two approximation methods applied to instance retrieval using the Gene Ontology. Section 2.6 concludes our work.

2.2 Instance Retrieval Queries

In this article we focus on the following instance retrieval problem:

Definition 1 (Instance retrieval w.r.t. some query) *Given an A-Box $\mathcal{A}$ and a query Q, i.e., a concept expression, find all individuals a such that a is an instance of Q, i.e., $\{a \,|\, \forall a \in \mathcal{A}, a : Q\}$.*

Often, an analogy is made between databases (DBs) and DL KBs. The schema of a DB corresponds to the T-Box and the DB instances correspond to the A-Box. However, A-Boxes have a very different semantics. This makes query answering in a DL setting often much more complex than query answering in a DB. Given the expressivity of DLs, retrieving instances to a query cannot simply be reduced to model checking as in the database framework because there is no single minimal model for a query. Knowledge Bases may contain nondeterminism and/or incompleteness. Therefore, deductive reasoning is needed when answering a query in a DL setting.

2.2.1 Conjunctive Queries

A-Box query languages have been quite weak for earlier DL systems. Usually they supported very simple A-Box queries like instantiation (is individual i an instance of concept C, i.e., $i : C$), realisation (what are the most specific concepts i is an instance of), and retrieval (which individuals are instances of concept C).

In [55] an approach for answering conjunctive queries over arbitrary DL KBs is given based on the translation of the query into an equivalent concept expression, i.e., by *rolling up* the query.

Definition 2 (Boolean Conjunctive Query) *A* Boolean conjunctive query Q *is of the form* $q_1 \wedge \cdots \wedge q_n$*, where* $q_1, \ldots, q_n$ *are query terms of the form* $x : C$ *or* $\langle x, y \rangle : R$*, where* C *is a concept,* R *is a role, and* x, y *are either individual names or variables.*

The approach makes use of the fact that binary relations in a conjunctive query can be translated into an existential restriction such that logical consequence is preserved. Standard DL inference methods can then be used to classify the concept expression the query is translated into as well as retrieve the instances that belong to it. The method of [55] enables us to use an expressive query language for arbitrary expressive DL KBs.

Because binary relations in a conjunctive query can be translated into an existential restriction such that logical consequence is preserved, standard DL inference methods can then be used to classify the concept expression the query is translated into as well as retrieve the instances that belong to it. [55] enables us to use an expressive query language for arbitrary expressive DL KBs.

2.2.2 Instance Store

DL reasoning is hard, especially in the case of instance retrieval when the number of instances grows very large. To speed up the overall cost of instance retrieval, one can address the number and cost of checking whether a single instance belongs to a query.

Instance Store [56] is developed to speed up instance retrieval by replacing costly instantiation checks $a : Q$ with database retrieval. However, Instance Store can not replace all DL reasoning steps using database retrieval. In some situations DL instantiation checks must still be performed. An analysis of the Instance Store revealed a drastic breakdown in performance in these situations, which hampers its goal to scale-up reasoning to ontologies with a large number of instances. At the moment Instance Store only supports role-free A-Boxes, i.e., relationships between instances in the A-Box are not allowed, but this was sufficient for our purpose.

Checking each individual in an A-Box if it instantiate a conjunctive query Q is inefficient especially for large A-Boxes. A technique that has been developed to scale DL reasoners for role-free A-Boxes with a large number of instances is the Instance Store (IS) [56]. The IS combines DL reasoning with Database retrieval to speedup the process of instance retrieval.

To describe the IS algorithm we use the following notation. For a T-Box $\mathcal{T}$, an A-Box $\mathcal{A}$, a concept C, and a query Q:

- $C\downarrow_{\mathcal{T}}$ stands for the set of atomic concepts in $\mathcal{T}$ subsumed by C (i.e., equivalents and descendants of C). The set of individuals in $\mathcal{A}$ that realise *some* concept in $Q\downarrow_{\mathcal{T}}$ is denoted by I_1. Any individual in I_1 is an answer to Q.

- $\lceil C\rceil_{\mathcal{T}}$ stands for the set of most specific concepts in $\mathcal{T}$ subsuming C. The set of individuals in $\mathcal{A}$ that realise *every* concept in $\lceil Q\rceil_{\mathcal{T}}$ is denoted by I_2. The individuals in I_2 must be checked for instanciation of Q constituting I_3.

Please note, that [56] showed that the I_1 and I_3 contains all answers to Q. Using this notation, IS can be described as a 7 step process:

1. use the DL reasoner to compute $Q\downarrow_{\mathcal{T}}$;
2. use the database to find the set of individuals I_1;
3. use the reasoner to check whether Q is equivalent to any atomic concept in $\mathcal{T}$; if that is the case then simply return I_1 and terminate;
4. otherwise, use the reasoner to compute $\lceil Q\rceil_{\mathcal{T}}$;
5. use the database to compute I_2;
6. use the 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;
7. return $I_1 \cup I_3$ and terminate.

Step 1 an 2 compute I_1. Step 3 and 4 are optimisations which avoid unnecessary computation of I_2 resp. I_3. Step 5 and 6 computes I_2 and I_3.

2.3 Approximation Techniques for Instance Retrieval

There are three components of the instance retrieval problem where approximation methods can be applied:

The Query. The query can be made weaker, i.e., more general, by omitting or replacing parts of the query. The underlying assumption is that simpler queries are easier to check.

The Ontology. We assume that the query is formulated relative to a given ontology. Concept expressions in the ontology (representing for example an instantiation check) can be approximated by weaker or stronger concept expressions.

The Instance Descriptions. In order to check whether instances belong to the query, first the descriptions of instances are translated into equivalent concept expressions. Consequently, those concept expressions can be approximated by weaker or stronger concept expressions.

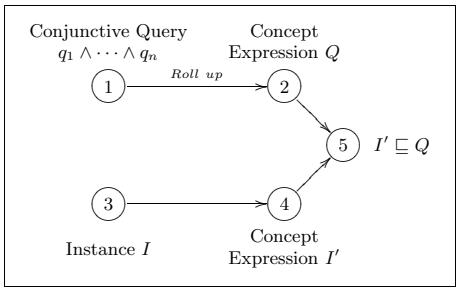


Figure 2.1: Various components

This section reviews the techniques of [84] and [93] that can be used to approximate instance retrieval in DL. Figure 2.1 gives an overview of the various components used in instance retrieval. The method of [84] was proposed to approximate satisfiability of concept expressions (usable in step 5 of Figure 2.1).[2] The method of [93] can be used to approximate conjunctive queries, or its concept expression counterpart (usable in steps 1 and 2 of Figure 2.1).

Both methods propose to approximate an instantiation test using a sequence of tests $C_1, \ldots, C_n$. Assuming that less complex tests can be answered in less time, instance checking can then be speeded up. However, both methods differ in their strategy for selecting the sequence of expressions C_i to be checked successively. In general, [93] argues that the order should balance two factors:

1. The *smoothness* of the approximation. In particular, the next test C_{i+1} should lead to the next best approximation.

2. The potential contribution of the extension of C_{i+1} to the *time complexity* of the tests to be done by the system.

2.3.1 Approximating Description Logic Satisfiability

In DLs, satisfiability checking can be seen as the most basic task as many reasoning services can be restated into satisfiability checks [4]. In [84] a technique has been developed to approximate satisfiability checks. Concept expressions are approximated by two sequences $C_1, \ldots, C_n$ of simpler concept expressions, obtained by syntactic manipulations, which can be used to determine the satisfiability of the original concept expression.

For every subconcept D, [84] defines the *depth* of D to be 'the number of universal quantifiers occurring in C and having D in its scope'. The scope of $\forall R.\phi$ is ϕ which can be any concept term containing D. A sequence of weaker (stronger) approximated concepts can be defined, denoted by $C_i^\top$ ($C_i^\bot$), by replacing every *existentially quantified*

[2] [84] should also be usable in steps 2 and 4, although not proposed originally.

subconcept, i.e., $\exists R.\phi$ where ϕ is any concept term, of depth greater or equal than i by $\top$ ($\bot$). Concept expressions are assumed to be in negated normal form (NNF) before approximating them.

Theorem 1 ([84]) *For each i, if $C_i^\top$ is unsatisfiable then $C_j^\top$ is unsatisfiable for all $j \geq i$, hence C is unsatisfiable. For each i, if $C_i^\bot$ is satisfiable then $C_j^\bot$ is satisfiable for all $j \geq i$, hence C is satisfiable.*

The sequences $C^\top$ and $C^\bot$ can be used to gradually approximate the satisfiability of a concept expression. [84] only replaces subconcepts $D \equiv \exists R.C$ as the worst case complexity depends on the nesting of existential and universal quantifiers. Theorem 1 leads to the following for $C^\bot$-approximation:

$$\begin{array}{llcll} (I \sqsubseteq Q)_i^\bot & \text{is not satisfiable} & \Leftrightarrow & (I \sqcap \neg Q)_i^\bot & \text{is satisfiable} \quad \Rightarrow \\ (I \sqcap \neg Q) & \text{is satisfiable} & \Leftrightarrow & (I \sqsubseteq Q) & \text{is not satisfiable} \end{array}$$

Therefore, we are only able to reduce complexity when approximated subsumption tests are not satisfiable. When an approximated subsumption test $(I \sqsubseteq Q)_i^\bot$ is satisfiable, nothing can be concluded and the approximation continues to level $i+1$ until no more approximation is applicable, i.e., the original concept term is obtained. Analogously, from Theorem 1 one obtains that when $(I \sqsubseteq Q)_i^\top$ is satisfiable this implies that $(I \sqsubseteq Q)$ is satisfiable. When $(I \sqsubseteq Q)_i^\top$ is not satisfiable nothing can be deduced and the approximation continues to level $i+1$.

Research on this kind of DL approximation is quite limited. [84] is the only method that deals with approximation of satisfiability in DLs. Few results have only been obtained recently [38].

2.3.2 Approximating Conjunctive Queries

In [93] a method is introduced for approximating conjunctive queries. The method computes a sequence $Q^1, \ldots, Q^n$ of queries such that: (1) $i < j \Rightarrow Q^i \sqsupseteq Q^j$ and (2) $Q^n \equiv Q$. The first property ensures that the quality of the results of the queries doesn't decrease. The second property ensures that the last query computed returns the desired exact result.

The proposed method can easily be adapted for instantiation checks. The computed sequence $Q^1, \ldots, Q^n$ is used to generate the sequence $C_1^\Delta, \ldots, C_n^\Delta$ with $C_i^\Delta = a : Q^i$. Assuming that less complex queries can be answered in less time, instantiation checks can then be speeded up using the following implication:

$$(I \not\sqsubseteq Q') \wedge (Q \sqsubseteq Q') \Rightarrow I \not\sqsubseteq Q$$

In [93] the sequence of subsuming queries $Q^1, \ldots, Q^n$ is constructed by stepwise adding a conjunct (of the original query) starting with the universal query.

A problem that remains to be solved in this approach is a strategy for selecting the sequence of queries to be checked successively. This problem boils down to ordering the conjuncts of the query which should balance the two factors 'smoothness' and 'time complexity'.

As described in [93] the smoothness of the approximation can be guaranteed by analyzing the dependencies between variables in the query. After translating the conjunctive query to a DL expression, these dependencies are reflected in the nesting of subexpressions. As the removal of conjuncts from a concept expression is equivalent to substitution by $\top$, this nesting provides us with a selection strategy to determine a sequence of approximations S_i where all subexpressions at depth greater or equal than i are replaced by $\top$. Hence, this method is somewhat similar to $C^{\top}$-approximation except that it is restricted to the conjunctive query, i.e., the instance description is not approximated, and it can replace any conjunct in the query with $\top$, not only existentially quantified conjucts.

Typically, however, queries often have a very flat structure. For example, all queries used in our experiments with the Gene Ontology are of depth one. This means that S_0 is the query $\top$ whereas S_1 is already the original query. To avoid this bad approximation scheme, next we propose an improved strategy.

An Improved Approximation Strategy

To overcome the flatness of queries typically found in ontologies, we propose a strategy that also provides an order for subexpressions at the same level of depth. A possible ordering is the expected time contribution of a conjunct to the costs of the subsumption test. As measuring the actual time is practically infeasible, a heuristic is proposed.

For this purpose, we unfold the conjuncts using the definitions of the concepts from the ontology occurring in the conjunct. In order to determine a suitable measure of complexity for expressions, we consider the standard proof procedure for DLs. Most existing DL reasoners are based on tableau methods, which determine the satisfiability of a concept expression by constructing a constraint system based on the structure of the expression. As the costs of checking the satisfiability of an expression depends on the size of the constraint system, we can use this size as a measure of complexity. As determining the exact size of the constraint system requires to run the tableau method, heuristics are used for estimating the size. Based on this estimated size, we determine the order in which conjuncts at the same level of depths are considered.

In the following, we propose a method for estimating the size of the tableau for expressions in $\mathcal{ALC}$ that will be used in the experiments. The tableau rules [4] provide us with quite a good idea about an estimation of the maximal size of the tableau in the worst case. For this purpose, we define a function Φ that assigns a natural number representing the estimated size of the corresponding constraint system to an arbitrary $\mathcal{ALC}$ expression

in the following way:

$$\begin{aligned}
\Phi(A) &= 1 \\
\Phi(\neg A) &= 0 \\
\Phi(C \sqcap D) &= 2 + \Phi(C) + \Phi(D) \\
\Phi(C \sqcup D) &= \phi + 2 + \Phi(C) + \Phi(D) \text{ where } \phi \text{ is the current value of } \Phi(E) \\
\Phi(\exists\ R.C) &= 2 + \Phi(C) \\
\Phi(\forall\ R.C) &= n + n \cdot \Phi(C) \text{ where } n \text{ is the number of existential quantifiers in } E
\end{aligned}$$

A **and** $\neg A$**:** Atomic concepts are added as a single constraint. Negated concepts are not added as they are merely used to check the existence of a contradiction.

$C \sqcap D$**:** Two new constraints are added. The expressions in these constraints have to be evaluated recursively, therefore, we also have to estimate the number of constraints that will be generated by C and D.

$C \sqcup D$**:** Two new constraints are added and each of the constraints has to be evaluated recursively, however, we have to deal with two separate constraint systems from this point on. The number of constraints in the system at this point has to be doubled. For an estimation we add the current estimation value.

$(\exists\ R.C)$**:** Two new constraints are added, one for the relation and one restricting the object in the relation to C Object y has to be evaluated recursively.

$(\forall\ R.C)$**:** A new constraint has to be added for every existing constraint xRy in the constraint system S and each one has to be evaluated recursively. As we do not know how many of these statements are or will be in S, we use the overall number of existential quantifiers in the expression that can lead to the addition of these constraints as an upper bound.

The value Φ can now be computed for each conjunct in the query and be used as a basis for determining the order in which conjuncts at the same level of nesting are processed.

2.4 Experimental Evaluation

In this section experimental results are shown of the approaches described in the previous section. The main question focused on in the experiments is *if*, and if yes, in *what way* does approximation reduce the complexity of the retrieval task. We focus on the number of operations needed and the overall computation time used. The goal of our approximation approach is to replace costly reasoning operations by a (small) number of cheaper approximate reasoning operations. The approximation methods used are sound and complete. Therefore, the suitability of the approximation methods depend solely on the time gained (or lost) when classical operations are replaced by a number of approximate ones.

Our experiments were made with the Gene ontology and Instance Store [56]. The focus of our experiments are those queries where Instance Store cannot replace all DL reasoning with database retrieval, but must still check the instantiations of some instances.

These instantiation checks were found to be a bottleneck in the scalability of this approach. We originally started with 17 queries (with $Q1$ to $Q6$ user formulated queries and queries $Q7$ to $Q17$ artificial), but discarded the queries that didn't require instantiation checks from further experiments.

Table 2.1: Performed Subsumption tests

	normal			$C^\top$			$C^\bot$			C^Δ		
		true	false		true	false		true	false		true	false
Q2										$L0$	20	0
										$L1$	20	0
				$L0$	0	19	$L0$	19	0	$L2$	9	11
	no	9	11	no	9	11	no	9	11	no	9	0
Q8										$L0$	607	0
				$L0$	0	606	$L0$	606	0	$L1$	10	597
	no	10	597	no	10	597	no	10	597	no	10	0
Q12				$L0$	0	7871	$L0$	7871	0	$L0$	15	7856
	no	15	7856	no	15	7856	no	15	7856	no	15	0
Q14										$L0$	408	0
										$L1$	5	403
				$L0$	0	407	$L0$	407	0	$L2$	5	0
	no	5	403	no	5	403	no	5	403	no	5	0
Q15				$L0$	0	6693	$L0$	6693	0	$L0$	6693	0
	no	46	6647	no	46	6647	no	46	6647	no	46	6647
Q17				$L0$	0	7873	$L0$	7873	0	$L0$	1	7872
	no	1	7872	no	1	7872	no	1	7872	no	1	0

The results of the first experiments are shown in Table 2.1, which is divided into four columns with each column reporting the number of subsumption tests performed. The first column reports results for the experiment without any approximation, the second column with $C^\top$-approximation, the third column with $C^\bot$-approximation, and the fourth column with C^Δ-approximation. Each column is further divided into smaller rows and columns. The rows represent the level of the approximation used, where *no* denotes without approximation, and Li denotes the level of the approximation approach. The subcolumns show the number of subsumption tests that resulted in true or false.[3] This distinction is important, because Section 2.3 tells us that only when a $C^\top$-approximated subsumption succeeds, or a $C^\bot$- or C^Δ-approximated subsumption test fails we obtain a reduction in complexity.

[3]We will use the shorthand 'true subsumption test' and 'false subsumption test' to indicate these two distinct results.

2.5 Discussion

Let us first focus on the question *if* the approximation methods can lead to any reduction in complexity. Table 2.1 shows that $C^{\top}$- and $C^{\bot}$-approximation cannot reduce the number of normal subsumption tests. Only C^{Δ} is able to reduce, except for $Q15$, all false subsumption tests to 0.

The first column in Table 2.1 shows that much more false subsumption tests are needed than true subsumption tests. This indicates that $C^{\top}$-approximation is wrong in this approach as it can only be used to lower the complexity of true subsumption tests, which is negligible when compared to false subsumption tests. This may explain its bad approximating behaviour, however, $C^{\bot}$ also performs badly, which does approximate false subsumption tests. Closer analysis shows that *term collapsing* [38], i.e., the substitution of terms by $\top$ or $\bot$ results in the query becoming equivalent to $\top$ or $\bot$, is the reason for this. An analysis of $C^{\bot}$ shows that this occurs in *all cases*.

Apart from looking at *if* an approximation method can successfully reduce the number of normal subsumption tests, we must also consider the cost for obtaining the reduction, i.e., in *what way* are the normal subsumption tests reduced. For example, approximating $Q8$ changes $607 = 10 + 597$ normal subsumption tests into 10 normal subsumption tests, 607 C_1^{Δ} subsumption tests, and 607 C_0^{Δ} subsumption tests. Thus, the number of subsumption tests may increase, but the complexity of most tests will be lower than normal. Note however, that some computations seem unnecessary as nothing can be deduced from them, e.g., the 607 C_0^{Δ} tests. Obviously, in this approach unnecessary subsumption tests should be minimized. Several cases can be observed in the experiments with C^{Δ}-approximation. Either no subsumption test is unnecessary ($Q12, Q17$), some subsumption tests are unnecessary ($Q2, Q8, Q14$), or all subsumption tests are unnecessary ($Q15$).

Figure 2.2: Time needed for Subsumption tests (in milliseconds)

	normal	$C^{\top}$	$C^{\bot}$	C^{Δ}
Q2	175	348	299	547
Q8	5373	8383	7753	9912
Q12	61410	93100	85764	56478
Q14	4372	6837	6017	7391
Q15	61560	90847	83714	114162
Q17	113289	158218	144689	93074

This distinction seems to influence the overall time needed when approximating a query. Table 2.2 reports the overall time in milliseconds needed for each query. For comparison $C^{\top}$ and $C^{\bot}$ are also reported. For queries having unnecessary subsumption tests, approximation always leads to more computation time. In those cases, reducing the complexity of subsumption tests do not weigh up to the costs of additional (unnecessary) subsumption tests. For queries having no unnecessary subsumption tests, approximation

does save time when compared to the normal case.

Another observation of Table 2.1 is that false subsumption tests for C^{Δ} only occur at *one level*. It seems that the conjunct that is added to the approximated conjunctive query on which the false subsumption tests occur is crucial in determining the outcome.The role of conjunct in a subsumption test is still unclear. More research is needed if this conjunct (or a group of conjuncts) can be identified in advance to speed up approximation.

2.6 Conclusions

Instance retrieval is one of the most important inferences in the Semantic Web. In order to make methods more scalable for ontologies with a large set of instances we investigated two approximation methods and evaluated them on a benchmark set. Both methods use a similar idea, i.e., removing parts of an expression to make it simpler to speed up retrieval. However, the method of [84] shows bad approximating behavior because the selection and substitution of subconcepts is too restrictive. The method of [93] was extended with a heuristic for subconcept selection and shows some potential for speeding up instance retrieval. However, more research is needed to improve the heuristic and to determine if the approximation method can be used to speed up instance retrieval.

Chapter 3

SCREECH – Faster OWL using split programs

by PASCAL HITZLER & DENNY VRANDECIC

We propose a new technique for approximate ABox reasoning with OWL DL ontologies. Essentially, we obtain substantially improved reasoning performance by disregarding non-Horn features of OWL DL. Our approach comes as a side-product of recent research results concerning a new transformation of OWL DL ontologies into negation-free disjunctive datalog [57, 58, 60, 72], and rests on the idea of performing standard resolution over disjunctive rules by treating them as if they were non-disjunctive ones. We analyze our reasoning approach by means of non-monotonic reasoning techniques, and present an implementation, called SCREECH.

This chapter is essentially a substantial update of Chapter 4 in Deliverable D2.1.2.

3.1 Introduction

Knowledge representation and reasoning on the Semantic Web is done by means of ontologies. While the quest for suitable ontology languages is still ongoing, OWL [104] has been established as a core standard. It comes in three flavors, as OWL Full, OWL DL and OWL Lite, where OWL Full contains OWL DL, which in turn contains OWL Lite. The latter two coincide semantically with certain description logics [4] and can thus be considered fragments of first-order predicate logic.

OWL ontologies can be understood to consist of two parts, one intensional, the other extensional. In description logics terminology, the intensional part consists of a TBox and an RBox, and contains knowledge about concepts (called *classes*) and the complex relations between them (called *roles*). The extensional part consists of an ABox, and contains knowledge about entities and how they relate to the classes and roles from the intensional part. For the Semantic Web, TBox and RBox shall provide background vocabulary,

while (annotated) webpages etc. constitute ABoxes which are interlinked with intensional knowledge. The Semantic Web thus envisions a distributed knowledge source, built from OWL ontologies and intertwining the knowledge like the World Wide Web interconnects websites.

With an estimated 25 million active websites today and correspondingly more webpages, it is apparent that reasoning on the Semantic Web will have to deal with very large ABoxes. Complexity of ABox reasoning — also called *data complexity* — thus measures complexity in terms of ABox size only, while considering the intensional part of the ontology to be of constant size. For the different OWL variants, data complexity is at least NP-hard, which indicates that it will not scale well in general [59]. Methods are therefore being sought to cope with large ABoxes in an approximate manner.

The approach which we propose is based on the fact that data complexity is polynomial for non-disjunctive datalog. We utilize recent research results [57, 58, 60, 72] which allow the transformation of OWL DL ontologies into disjunctive datalog. Rather than doing (expensive) exact reasoning over the resulting disjunctive datalog knowledge base, we do approximate reasoning by treating disjunctive rules as if they were non-disjunctive ones. The resulting reasoning procedure is complete, but may be unsound in cases. Its data complexity is polynomial. We are also able to give a characterization of the resulting approximate inference by means of standard methods from logic programming semantics.

This chapter is structured as follows. In Section 3.2, we first discuss the general rationale behind approximate reasoning, and how it relates to other reasoning frameworks. We then recall formal terminology and notation for OWL DL, and shortly review datalog and SLD-resolution. Then, in Section 3.4, we explain how OWL DL ontologies can be transformed into disjunctive datalog. In Section 3.5 we introduce the new approximate SLD-resolution procedure which we propose. The presentation of our implementation SCREECH in Section 3.6 is followed by an Example in Section 3.7, and an experimental evaluation in Section 3.8. We conclude and discuss future work in Section 3.9.

3.2 Non-Classical Reasoning — Common Grounds

Classical logic – a term which encompasses mainly propositional and first-order predicate logic – is the foundation for many knowledge representation and reasoning paradigms in artificial intelligence and related areas, such as semantic web. However, reasoning in these paradigms is often non-classical, i.e. it is obtained by modifying classical logic. Some of these modifications are syntactical. Conceptually, however, semantic differences are more important. From a bird's eye perspective, these semantic differences can often be perceived as a modification of the notion of model, which in the usual way has impact on the inference relation considered. We will elaborate a bit on this.

A semantic perspective on approximate reasoning is depicted in Figure 3.1. When a theory is being considered, classical reasoning may be of high computational complexity

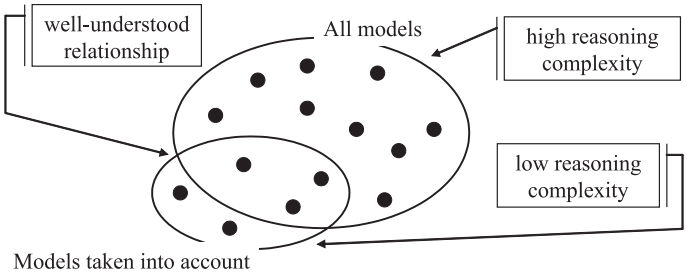


Figure 3.1: Semantic view on approximate reasoning

and thus be unsuitable for time-critical tasks. By taking different models into account than the classical ones, the complexity of reasoning can be reduced. The resulting approximate inference may be incomplete or unsound with respect to classical inference, but in a controlled and well-understood manner, which makes the inferences suitable for further use.

Similar situations occur in the context of other sophisticated reasoning techniques. For non-monotonic reasoning, for example, a subset of the classical models is usually considered, which is selected by means of e.g. additional syntax constructs or by redefining the semantics of existing ones. Non-monotonic reasoning thus allows to arrive at conclusions which cannot be derived using classical reasoning: It is complete, but unsound, and can be described as *supraclassical* [70]. The rationale in this case is to model aspects of human commonsense reasoning like *jumping to conclusions*, again in a controlled and well-understood manner. Complexity considerations are often treated as secondary in this context.

Paraconsistent reasoning — or reasoning with inconsistency — can be approached from a similar perspective. While inconsistent knowledge bases have no classical models, paraconsistent reasoning strives to identify suitable models to be assigned to the knowledge base nevertheless, in order to allow the inference of meaningful consequences. As such, paraconsistent reasoning is sound, but incomplete with respect to classical logic, and can thus be termed *subclassical*.

reasoning approach	focus	models taken into account	typical complexity
classical		all classical models	high
non-monotonic	commonsense	some classical models	very high
paraconsistent	inconsistency	more than the classical models	high
approximate	performance	variable	low

Table 3.1: Comparision of non-classical reasoning approaches

Table 3.1 summarizes our discussion. While the table can certainly be extended further taking other forms of reasoning into account, we restrict ourselves to the mentioned

examples, as the main goal of this chapter is to present an approximate reasoning method for OWL DL, and not a comparative theory of reasoning approaches. We have included this discussion because it explains the general rationale behind our approximate reasoning method, and will help us in analyzing it. Indeed, in all reasoning paradigms mentioned, it is important to obtain a clear understanding of the inference relation computed. This can be done by semantic analyses, i.e. by characterizations of the models taken into account. From the general perspective described in this section, it will later come as no surprise to the reader that we will analyze our approximate reasoning methods by means of standard techniques from non-monotonic reasoning. Indeed, in our particular case the models taken into account for approximate reasoning will turn out to be a subset of the classical models, as in non-monotonic reasoning.

3.3 Preliminaries

3.3.1 OWL DL Syntax and Semantics

OWL DL is a syntactic variant of the $\mathcal{SHOIN}(\mathbf{D})$ description logic [52]. Hence, although several XML and RDF syntaxes for OWL DL exist, it will be convenient to use the traditional description logic notation since it is more compact, and we recall the notation below. For the correspondence between this notation and various OWL DL syntaxes, see [52].

We indeed assume that the reader is familiar with OWL and thus with $\mathcal{SHOIN}(\mathbf{D})$, as space restrictions forbid to reintroduce them, but recall that $\mathcal{SHOIN}(\mathbf{D})$ supports reasoning with concrete datatypes, such as strings or integers [68]. Recall also that the description logic syntax for concepts in $\mathcal{SHOIN}(\mathbf{D})$ is defined as follows, where A is an atomic concept, R is an abstract role, S is an abstract simple role, $T_{(i)}$ are concrete roles, d is a concrete domain predicate, a_i and c_i are abstract and concrete individuals, respectively, and n is a non-negative integer:

$$\begin{array}{rcl} C & \rightarrow & A \mid \neg C \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \exists R.C \mid \forall R.C \mid \geq n\, S \mid \leq n\, S \mid \{a_1, \ldots, a_n\} \mid \\ & & \mid \geq n\, T \mid \leq n\, T \mid \exists T_1, \ldots, T_n.D \mid \forall T_1, \ldots, T_n.D \\ D & \rightarrow & d \mid \{c_1, \ldots, c_n\} \end{array}$$

The $\mathcal{SHIQ}(\mathbf{D})$ description logic is obtained from $\mathcal{SHOIN}(\mathbf{D})$ by disallowing nominal concepts of the form $\{a_1, \ldots, a_n\}$ and $\{c_1, \ldots, c_n\}$, and by allowing qualified number restrictions of the form $\geq n\, S.C$ and $\leq n\, S.C$, for C a $\mathcal{SHIQ}(\mathbf{D})$ concept and S a simple role.

As description logics, $\mathcal{SHOIN}(\mathbf{D})$, i.e. OWL DL, and $\mathcal{SHIQ}(\mathbf{D})$ inherit their semantics from first-order logic by the standard translations known e.g. from [54], which we do not repeat here.

3.3.2 Datalog and SLD-Resolution

A (*definite* or *negation-free*) *disjunctive logic program* P consists of a finite set of *clauses* or *rules* of the form

$$\forall x_1 \ldots \forall x_n.(H_1 \vee \cdots \vee H_m \leftarrow A_1 \wedge \cdots \wedge A_k),$$

commonly written as

$$H_1 \vee \cdots \vee H_m \leftarrow A_1, \ldots, A_k,$$

where $x_1, \ldots, x_n$ are exactly all variables occuring in $H_1 \vee \cdots \vee H_m \leftarrow A_1 \wedge \cdots \wedge A_k$, and all H_i and A_j are atoms over some given first-order language Σ. The disjunction $H_1 \vee \cdots \vee H_m$ is called the *rule head*, and the conjunction $A_1 \wedge \cdots \wedge A_k$ is called the *rule body*. The set of all ground instances of atoms defined over Σ is called the *Herbrand base* of P and is denoted by B_P. The set of all ground instances of rules in P is denoted by $\mathsf{ground}(P)$. A rule is said to be *non-disjunctive* if $m = 1$. It is called a *fact* if $k = 0$. We abstract from the order of the atoms in the heads respectively bodies; it is not important for our results. A disjunctive logic program is called a *(disjunctive) datalog* program if it does not contain function symbols.

Note that we do not consider logic programs to come with one specific semantics. Some people for example associate datalog with the minimal model semantics only. For our treatment, datalog and logic programs are defined via syntax only. We do not specify a specific semantics because in the following we will discuss *different* semantics for logic programs in their relation to proof procedures. One of the semantics we will consider is the semantics coming from interpreting logic programs as a set of first order formulas, and in this case we use $\models$ to denote entailment in classical first-order predicate logic.

SLD-resolution (see e.g. [66]) is an efficient top-down query-answering technique for programs consisting of non-disjunctive rules, and has been implemented and successfully applied in standard Prolog systems.[1] In this framework, a ground atom can be derived from a program if and only if it is true in the least (and thus in all) Herbrand models of the program.

In the following, we mean by a *conjunctive query* simply a conjunction $B_1 \wedge \cdots \wedge B_n$ of atoms. The query is called *ground* if it does not contain any variables.

Given a conjunctive query $B_1 \wedge \cdots \wedge B_n$, an *SLD-resolution step* on the atom B_i with a non-disjunctive rule $H \leftarrow A_1, \ldots, A_k$ produces a conjunctive query

$$B_1\theta \wedge \cdots \wedge B_{i-1}\theta \wedge A_1\theta \wedge \cdots \wedge A_k\theta \wedge B_{i+1}\theta \wedge \cdots \wedge B_n\theta$$

where θ is the most general unifier of B_i and H. An *SLD-refutation* of a conjunctive query $B_1 \wedge \cdots \wedge B_n$ in a non-disjunctive program P is a finite sequence of conjunctive queries $Q_0, \ldots, Q_n$, where (*i*) $Q_0 = B_1 \wedge \cdots \wedge B_n$, (*ii*) each Q_i with $i > 0$ is obtained from Q_{i-1} by an SLD-resolution step with some rule from P on some literal B_i, and (*iii*)

[1] Like SWI or XSB Prolog, http://www.swi-prolog.org, http://xsb.sourceforge.net.

$Q_n = \Box$, i.e. the conjunctive query Q_n does not contain any literals. If an SLD-refutation of $B_1 \wedge \cdots \wedge B_n$ in P exists, we write $P \vdash B_1 \wedge \cdots \wedge B_n$.

One of the fundamental results in logic programming states that $A \in B_P$ can be proven by SLD-resolution if and only if A is a logical consequence of P, i.e. if and only if A is true in the least Herbrand model of P:

Theorem 2 ([66]) [2] *For a ground conjunctive query $B_1 \wedge \cdots \wedge B_n$ and a non-disjunctive program P, $P \vdash B_1 \wedge \cdots \wedge B_n$ if and only if $P \models B_1 \wedge \cdots \wedge B_n$. In other words, entailment of ground conjunctive queries under SLD-resolution is entailment in predicate logic.*

SLD-resolution also allows deriving answers to non-ground queries: For a conjunctive (and not necessarily ground) query Q there exist an SLD-refutation if and only if $P \models \exists x_1 \ldots \exists x_n.Q$, where $x_1, \ldots, x_n$ are the variables occuring in Q. By keeping track of the most general unifiers used in the process, it is also possible to obtain bindings for (some of) the x_i in the form of (answer) substitutions θ, such that $P \models \exists y_1 \ldots \exists y_k(Q\theta)$, where the y_i are exactly those variables occurring in $Q\theta$. In order to keep our exhibition focused, we will only deal with ground queries.

3.4 Reducing OWL DL Knowledge Bases to Disjunctive Datalog Programs

We utilise recent research results about the transformation of OWL DL ontologies into disjunctive datalog, and perform approximate reasoning by transforming the disjunctive database into a non-disjunctive one. The transformation is based on the fact that OWL DL is a subset of first-order logic. OWL axioms can thus be translated directly into logical formulas and transformed into clausal form using any of the standard algorithms. The resulting clauses can be represented as disjunctive datalog rules which do not contain negation.

Note, however, that due to possible skolemization steps in the clausal form translation, the resulting datalog rules may contain function symbols. In general, datalog with function symbols is undecidable, but since we obtain the datalog program by a translation from OWL DL, which is decidable, inferencing over the resulting program must be decidable. Standard datalog engines, however, do in general not terminate in the presence of function symbols. To cope with this problem, a sophisticated method has been presented in [58, 60] which allows to get rid of the function symbols without loosing ABox consequences. As a result, we obtain a function- and negation-free disjunctive datalog program, which can be dealt with using standard techniques.

There is one other catch: The approach presented in [58, 60] does not yet allow to deal with nominals, i.e. it supports only $\mathcal{SHIQ}(\mathbf{D})$ instead of $\mathcal{SHOIN}(\mathbf{D})$ (the latter is

[2] Please note that this definition is identical to Definition 2 in Section 2.2.1

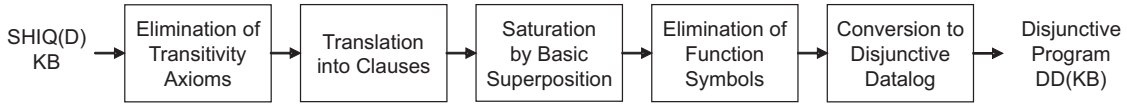


Figure 3.2: Algorithm for Reducing $\mathcal{SHIQ}(\mathbf{D})$ to Datalog Programs

the description logic coinciding with OWL DL). We remark that to date — and to the best of our knowledge — no reasoning algorithms for $\mathcal{SHOIN}(\mathbf{D})$ have been implemented. We will return to a possible treatment of nominals in our approach later.

The translation algorithm is schematically depicted in Figure 3.2. It transforms a $\mathcal{SHIQ}(\mathbf{D})$ knowledge base KB into a disjunctive datalog program DD(KB). The steps of the algorithm are as follows. (1) Transitivity axioms are removed by adding axioms of a form similar to $\forall S.C \sqsubseteq \forall S.(\forall S.C)$ for transitive roles S. (2) The knowledge base is translated into clausal form by standard transformations based on first-order predicate logic. This introduces function symbols due to necessary skolemization steps. (3) The TBox of the knowledge base is partially saturated by adding logical consequences. This is the crucial step of the algorithm. (4) The saturation from step (3) now allows to remove all function symbols which were introduced in step (2). Some additional axioms are added to ensure that the algorithm remains sound and complete. (5) The knowledge base is translated into disjunctive datalog clauses; this step is now straightforward.

It shall be noted that the details of the crucial step (3) are very sophisticated. They guarantee that the removal of function symbols in step (4) is at all possible. Step (3) is of exponential complexity, however for the ABox reasoning task which we focus on in this chapter, Step (3) can in principle be performed offline, as this step is independent of the ABox – but note that this offline computation may still be difficult if the TBox is large, which is a separate issue and deserves further in-depth studies which are outside the scope of this chapter. A full presentation of the translation with correctness proofs is technically involved and lengthy, and space restrictions forbid to go into further detail; we refer the interested reader to [58, 60]. In [57] full proofs are given which show amongst other things that KB is unsatisfiable if and only if DD(KB) is unsatisfiable. This suffices for reasoning over KB as reasoning tasks can be transformed into unsatisfiability checks.

3.5 Approximate Resolution

While approximate reasoning methods for propositional and first-order logic have been proposed (see e.g. [86, 84, 20, 17, 100, 37]), they have hardly been applied in the context of Semantic Web technologies. The few exceptions are reported e.g. in [93, 56, 38] — to the best of our knowledge, this list is exhaustive. The success of the approaches is mixed. [38] reports on an analysis indicating that straightforward adaptations of methods proposed by [84] do not suffice. [56] reports good results but is not an approximate

reasoning method in the more narrow sense as the reasoning performed is exact, and thus does not address the complexity problems underlying OWL DL reasoning. [93] deals with approximating queries, while we focus on ABox reasoning. We will now present a novel approach based on the translation of OWL DL to disjunctive datalog, as presented earlier.

3.5.1 Approximate SLD-Resolution

Having obtained the translated knowledge base in the form of a disjunctive datalog program, ABox reasoning remains NP-hard, and thus untractable. If the datalog program is non-disjunctive, though, reasoning is polynomial in the size of the ABox. We therefore propose the following approximate reasoning technique in order to facilitate this insight. Given a conjunctive query $B_1 \wedge \cdots \wedge B_n$, an *approximate SLD-resolution step* on the atom B_i with a disjunctive rule $H_1 \vee \cdots \vee H_m \leftarrow A_1, \ldots, A_k$ is a conjunctive query

$$B_1\theta \wedge \cdots \wedge B_{i-1}\theta \wedge A_1\theta \wedge \cdots \wedge A_k\theta \wedge B_{i+1}\theta \wedge \cdots \wedge B_n\theta$$

such that θ is the most general unifier of B_i and some H_j. *Approximate SLD-refutation* is defined analogously to SLD-refutation, where approximate SLD-resolution steps are used instead of (usual) SLD-resolution steps.

It is necessary to pursue the question what notion of entailment underlies the approximate reasoning technique we propose. Following the spirit of the observations from Section 3.2, we want to identify the set of models which underly the inference relation provided by approximate SLD-resolution. For this purpose, we need the following notion, which is derived from standard notions in non-monotonic reasoning over logic programs.

Definition 3 (cf. [3, 26, 47]) *A model M of a disjunctive program P is called* well-supported *if there exists a function $l : B_P \rightarrow \mathbb{N}$ such that for each $A \in M$ there exists a rule $A \vee H_1 \vee \cdots \vee H_m \leftarrow A_1, \ldots, A_k$ in $\mathsf{ground}(P)$ with $M \models A_i$ and $l(A) > l(A_i)$ for all i and k.*

Definition 3 is a straightforward adaptation of the notion of well-supported model for non-disjunctive programs, as given in [26]. For non-disjunctive (and negation-free) programs, the well-supported models are exactly the minimal ones, but this is not in general the case for disjunctive programs: Just consider the program consisting of the single rule $p \vee q \leftarrow$. Then $\{p, q\}$ is a well-supported model, but is not minimal.

Lifted appropriately to (non-disjunctive) programs with negation, the well-supported models coincide with the well-known stable models. This was shown in [26] and studied in-depth in [47, 45]. Stable models [32] provide the base for the most popular non-monotonic reasoning paradigm called *Answer Set Programming*, of which the two most prominent implementations are DLV and SMODELS [25, 90]. Our results thus stand within this well-established tradition.

It is apparent that $A \in B_P$ is entailed by a (disjunctive) program P by approximate SLD-resolution if and only if it is true in at least one well-supported model of P. This is called *brave reasoning with well-supported models*. A formal proof of the following proposition is omitted for space restrictions.

Proposition 1 *Entailment of ground conjunctive queries under approximate SLD-resolution is brave reasoning with well-supported models.*

As an example, consider the (propositional) program consisting of the two rules $p \vee q \leftarrow$ and $r \leftarrow p \wedge q$. Its minimal models are $\{q\}$ and $\{p\}$, so r is not bravely entailed by reasoning with minimal models. However all of $\{q\}$, $\{p\}$, $\{p, q\}$ and $\{p, q, r\}$ are well-supported models, so r is bravely entailed by reasoning with well-supported models.

There is an alternative way of formalizing approximate SLD-resolution using a modified notion of *split program* [83]. Given a rule

$$H_1 \vee \cdots \vee H_m \leftarrow A_1, \ldots, A_k,$$

the *derived split rules* are defined as:

$$H_1 \leftarrow A_1, \ldots, A_k \qquad \ldots \qquad H_m \leftarrow A_1, \ldots, A_k.$$

For a given disjunctive program P its *split program* P' is defined as the collection of all split rules derived from rules in P. Approximate SLD-resolution on P is obviously identical to SLD-resolution over P'.

Minimal models are well-supported, as can be seen from the following result which was obtained along the lines of research laid out in [47, 45].

Theorem 3 ([46]) *Let P be a disjunctive program. Then a model M of P is a minimal model of P if and only if there exists a function $l : B_P \rightarrow \mathbb{N}$ such that for each A which is true in M there exists a rule $A \vee H_1 \vee \cdots \vee H_m \leftarrow A_1, \ldots, A_k$ in $\mathsf{ground}(P)$ with $M \models A_i$, $M \not\models H_k$ and $l(A) > l(A_i)$ for all i and k.*

We hence have the following result, noting that $P \models Q$ for any ground conjunctive query Q and program P if and only if Q is true in all minimal models of P.

Proposition 2 *Let P be a (possibly disjunctive) program and Q be a ground conjunctive query with $P \models Q$. Then there exists an approximate SLD-refutation for Q.*

We remark that for negation-free disjunctive programs minimal models again coincide with *answer sets* [32], as in the currently evolving *Answer Set Programming Systems*, as already mentioned.

3.5.2 Approximate Resolution for OWL DL

Our proposal is based on the idea of converting a given OWL DL knowledge base into a function-free definite disjunctive logic program, and then to apply approximate resolution for ABox reasoning.

In order to be able to deal with all of OWL DL, we need to add a preprocessing step to get rid of nominals, i.e. we need to compile $\mathcal{SHOIN}(\mathbf{D})$ ontologies to $\mathcal{SHIQ}(\mathbf{D})$. We can do this by *Language Weakening* as follows: For every occurrence of $\{o_1, \ldots, o_n\}$, where $n \in \mathbb{N}$ and the o_i are abstract or concrete individuals, replace $\{o_1, \ldots, o_n\}$ by some new concept name D, and add ABox assertions $D(o_1), \ldots, D(o_n)$ to the knowledge base. Note that the transformation just given does in general not yield a logically equivalent knowledge base, so some information is lost in the process. Putting all the pieces together, we propose the following subsequent steps for approximate ABox reasoning for OWL DL.

1. Apply Language Weakening as just mentioned in order to obtain a $\mathcal{SHIQ}(\mathbf{D})$ knowledge base.
2. Apply transformations as in Section 3.4 in order to obtain a negation-free disjunctive datalog program.
3. Apply approximate SLD-resolution for query-answering.

The first two steps can be considered to be preprocessing steps for setting up the intensional part of the database. ABox reasoning is then done in the last step. From our discussions, we can conclude the following properties of approximate ABox reasoning for $\mathcal{SHIQ}(\mathbf{D})$.

- It is complete with respect to first-order predicate logic semantics.
- It is sound and complete wrt. brave reasoning with well-supported models.
- Data complexity of our approach is polynomial.

3.6 SCREECH OWL

A preliminary implementation of our approach is available as theSCREECH OWL approximatereasoner.[3]It is part of the KAON2 OWLtools.[4] KAON2[5] is the KArlsruhe ONtologyframework, which includes a fast OWL reasoner based on the transformationalgorithms mentioned in Section 3.4, and also includes many otherfeatures helpful to work

[3]http://logic.aifb.uni-karlsruhe.de/screech
[4]http://www.aifb.uni-karlsruhe.de/WBS/dvr/owltools
[5]http://kaon2.semanticweb.org

$$\begin{aligned}
\text{serbian} \sqcup \text{croatian} &\sqsubseteq \text{european} \\
\text{eucitizen} &\sqsubseteq \text{european} \\
\text{german} \sqcup \text{french} \sqcup \text{beneluxian} &\sqsubseteq \text{eucitizen} \\
\text{beneluxian} &\equiv \text{luxembourgian} \sqcup \text{dutch} \sqcup \text{belgian}
\end{aligned}$$

serbian(ljiljana)	serbian(nenad)	german(pascal)	french(julien)
croatian(boris)	german(markus)	german(stephan)	croatian(denny)
indian(sudhir)	belgian(saartje)	german(rudi)	german(york)

Figure 3.3: Example ontology

with ontologies. Among the KAON2 OWL tools, `deo` performs the language weakening step described in Section3.5.2 in order to obtain a $\mathcal{SHIQ}(\mathbf{D})$ knowledge base. As KAON2implements the sophisticated translation algorithms described in Section3.4, we can convert an OWL ontology into adisjunctive datalog program, e.g. by using the `dlpconvert` KAON2 OWLtool with the `-x` switch.SCREECH then accesses the results of the translation through theKAON2 API, creates the corresponding split programs and serializes them asHorn logic programs in Edinburgh Prolog syntax. The result canbe fed to any Prolog interpreter — or other logic programming engine —,which in turn can be used to perform ABox reasoning and inferencing over theknowledge base.For completeness, we need to mention that in general support for concretedomains and other features like integrity constraints is not necessarilyimplemented in off-the-shelf logic programming systems. In these cases,concrete domains etc. cannot be used. The KAON2 OWL tool`ded`,[3] for example, performs a language weakening step by removingall concrete domains, and may come in handy in such situations.

3.7 An Example

We demonstrate our approach by means of a simple OWL DL ontology. It contains only a class hierarchy and an ABox, and no roles, but this will suffice to display the main issues.

The ontology is shown in Figure 3.3, and its intended meaning is self-explanatory. Note that the fourth line,

$$\text{beneluxian} \equiv \text{luxembourgian} \sqcup \text{dutch} \sqcup \text{belgian},$$

translates into the four clauses

$$\begin{aligned} \text{luxembourgian}(x) \vee \text{dutch}(x) \vee \text{belgian}(x) &\leftarrow \text{beneluxian}(x), \\ \text{beneluxian}(x) &\leftarrow \text{luxembourgian}(x), \\ \text{beneluxian}(x) &\leftarrow \text{dutch}, \\ \text{and} \qquad \text{beneluxian}(x) &\leftarrow \text{belgian}(x). \end{aligned} \tag{3.1}$$

Thus, our approach changes the ontology by treating the disjunctions in line (3.1) as conjunctions. This change affects the soundness of the reasoning procedure. However, most of the ABox consequences which can be derived by approximate SLD-resolution are still correct. Indeed, there are only two derivable facts which do not follow from the knowledge base by classical reasoning, namely

$$\text{dutch}(\text{saartje}) \qquad \text{and} \qquad \text{luxemburgian}(\text{saartje}).$$

All other derivable facts are correct.

SCREECH translates the ontology from Figure 3.3 into the Prolog program listed in Figure 3.4. As standard implementations of SLD-resolution do not use fair selection functions and also use depth-first search for higher efficiency, they may sometimes fail to produce answers because they run into infinite branches of the search tree. This occurs, for example, when using SWI-Prolog[6]. A reordering of the clauses may improve the results, but does not solve the problem entirely. More satisfactory performance can be obtained by using SLD-resolution with tabling, as implemented e.g. in the XSB Prolog system[7]. In this case, all desired consequences can be derived.

3.8 Experiments and Evaluation

An approximate reasoning procedure needs to be evaluated on real data from practical applications. Handcrafted examples are of only limited use as the applicability of approximate methods depends on the structure inherent in the experimental data.

For our evaluation we have performed experiments with the OWL DL version of the GALEN Upper Ontology,[8] as it appears to be sufficiently natural and realistic. As it is a TBox ontology only, we populated GALEN's 175 classes randomly with 500 individuals.[9] GALEN does not contain nominals or concrete domains. GALEN has 673 axioms (the population added another 500). The TBox translation to disjunctive datalog took about 2300 ms, after which we obtained 2687 disjunctive datalog rules containing 267 disjunctions within 133 rules. Among these were 152 integrity constraints (i.e. rules with

[6] http://www.swi-prolog.org/
[7] http://xsb.sourceforge.net
[8] http://www.cs.man.ac.uk/∼rector/ontologies/simple-top-bio/
[9] Using the pop KAON2 OWL tool.

```
serbian(ljiljana).  serbian(nenad).    german(pascal).
french(julien).     croatian(boris).   german(markus).
german(stephan).    croatian(denny).   indian(sudhir).
belgian(saartje).   german(rudi).      german(york).
european(X)         :- serbian(X).
european(X)         :- croatian(X).
european(X)         :- eucitizen(X).
eucitizen(X)        :- german(X).
eucitizen(X)        :- french(X).
eucitizen(X)        :- beneluxian(X).
beneluxian(X)       :- luxembourgian(X).
beneluxian(X)       :- dutch(X).
beneluxian(X)       :- belgian(X).
dutch(X)            :- beneluxian(X).
luxembourgian(X)    :- beneluxian(X).
belgian(X)          :- beneluxian(X).
```

Figure 3.4: Example SCREECH output

empty head), which we removed for our experiment as they led to inconsistency of the database.[10] After splitting disjunctive rules, we arrived at 2802 Horn rules.

We then randomly selected classes and queried for their extension using the KAON2 datalog engine, both for processing the disjunctive datalog program and for the split program. Some of the typical results are listed in Table 3.2, which indicates a significant speed-up of about 40% on average, while the vast majority of the retrieved answers is correct. In a complete run we queried for the extensions of all 175 GALEN classes, resulting in a total number of 5809 classifications performed by SCREECH, of which 5353 (i.e. 92.2%) were correct. For 138 out of 175 classes the extension computed by SCREECH was correct. The average time saved when computing the extension was 38.0% over all 175 classes. Note that we obtain significant speed-up although the KAON2 datalog engine is not optimized for Horn programs, but rather tuned to efficient performance on definite disjunctive datalog.

The times were obtained with initial Java VM memory set to 256 MByte. Under memory restrictions, the speed-up is more significant, which is probably caused by the necessity to allocate additional memory for the DD reasoning task. Corresponding figures are given in Table 3.3. Our experiments also indicate that SCREECH may be useful when hardware is limited, for example in portable devices.

[10] This is an expected effect. Removal of the integrity constraints does not destroy completeness of the approximate reasoning procedure.

Time (DD)	Time (SPLIT)	Instances	Class Name
11036 ms	6489 ms	154/154	Biological_object
11026 ms	5959 ms	9/9	Specified_set
11006 ms	6219 ms	9/13	Multiple
11015 ms	5898 ms	16/16	Probe_structural_part_of_heart
11036 ms	7711 ms	4/4	Human_red_blood_cell_mature
11055 ms	5949 ms	24/58	Biological_object_that...

Table 3.2: Performance comparison for instance retrieval using disjunctive datalog (DD) vs. the corresponding split program (SPLIT), on the KAON2 datalog engine. *Instances* indicates the number of instances retrieved using DD versus SPLIT, e.g. class *Multiple* contained 9 individuals, while the split program allowed to retrieve 13 (i.e. the 9 correct individuals plus 4 incorrect ones). The full name of the class in the last row is Biological_object_that_has_left_right_symmetry.

3.9 Conclusions and Further Work

In a nutshell, our proposed procedure approximates reasoning by disregarding non-Horn features of OWL DL ontologies. We argue that this is a reasonable approach to approximate reasoning with OWL DL in particular because many of the currently existing ontologies rarely use language constructs that do not fall into the Horn fragment of OWL DL [103]. So it can be projected that even in the future these constructs will play a minor role and thus should be the first to be tempered with in order to gain tractable reasoning.

Our approach provides ABox reasoning with polynomial time complexity. While it is complete, it is also unsound with respect to first-order logic. We have shown, however, that the inference underlying our approach can be characterized using standard methods from the area of non-monotonic reasoning. We have also presented our implementation SCREECH, and verified the usefulness of our approach by means of experiments.

The checking whether a conjunctive query is a predicate logic consequence of a (negation-free) disjunctive logic program P amounts to checking whether the query is

Time (DD)	Time (SPLIT)	Instances	Class Name
32997 ms	4817 ms	154/154	Biological_object
33028 ms	4947 ms	9/9	Specified_set
32927 ms	4987 ms	9/13	Multiple
32977 ms	4957 ms	16/16	Probe_structural_part_of_heart
32987 ms	7350 ms	4/4	Human_red_blood_cell_mature
32947 ms	4796 ms	24/58	Biological_object_that...

Table 3.3: Performance comparison as in Table 3.2, but with 128 MByte intial memory.

valid in *all* minimal models of P, i.e. corresponds to *cautious* reasoning with minimal models. Theorem 3 suggests how an anytime algorithm for this might be obtained: After performing approximate SLD-resolution, it remains to be checked whether there is any (ground instance of a) rule used in the refutation of the query, which has an atom A in its head besides the one used in the refutation and such that A is (cautiously) entailed by the program. Such an algorithm might then first find a brave proof of a query, and then substantiate this proof by subsequent calculations. Our approach may also be useful for the quick derivation of *possible* answers to a query, which may then be used for efficient guidance of the search within a sound and complete OWL reasoner. These and other issues are currently under investigation.

Chapter 4

Robust Query Processing for Personalized Information Access

by PETER DOLOG, HEINER STUCKENSCHMIDT & HOLGER WACHE

4.1 Motivation

Users are often not able to formulate queries correctly which results in user dissatisfaction and frustration. This is even more the case for semantic web systems based on RDF for the following reasons:

- The data accessed often comes from different sources. The internal structure of these sources is not always known.
- The data is semi structured. Sources do not have to describe all aspects of the information resources.
- There is no fixed integrated schema. Each source can have its own schema, sources may make partial use of different available schemas.

With the increasing popularity of RDF as a representation language in domains such as medicine [94] or e-leaning [21] this problem becomes more pressing. If RDF query languages are to be used in a large scale we have to make sure that people will be able to formulate meaningful queries. If this is not the case, we have to find ways to still provide the user with the intended results.

Research in Cooperative Query answering is triggered by the observation that users are often not able to correctly formulate queries to databases that return the intended result. Cooperative query processing supports the user by automatically modifying the query in order to better fit the real intention of the user. Based on the assumed kind of mismatch

between the users intention and the formulated query there are different techniques used. We consider two basic mechanisms of cooperative query processing, *query refinement* and *query relaxation* which are briefly presented in the following.

4.1.1 Query Refinement and Relaxation

Due to a lack of knowledge of the contents and the structure of a database, users will often only be able to provide very broad queries, for example in terms of the type of the objects she wants to retrieve and maybe one or two properties. Taking an example from the domain of e-learning, the user might be able to specify that she is looking for a lecture on the Java Programming Language. Learning resources, however, are often annotated with a fair amount of metadata that specifies important information such as the assumed level of expertise and required previous knowledge. In order to select learning resources that are suited for the user, these additional properties have to be specified in the query as well. Dolog et al [21] show that this information can be included into a user query based on a user profile. They describe a method for automatically refining queries with information from the user profile thereby enabling a pre-selection of query answers.

A problem of the automatic refinement of queries lies in the fact that it often overshoots the target instead of too many results an automatically refined query often returns no result at all, because none of the resources exactly matches the users needs. A possible solution to this problem is to successively relax the constraints imposed in the refinement step. Different Techniques for relaxing queries have been proposed in the database area. Gaasterland et al [28] provide a unifying view on different relaxation techniques in terms of replacing subexpressions in the query. In other work we described an approach for relaxing conjunctive queries over description logic knowledge bases by removing conjuncts from the query in a particular order (see 2 and [93, 105]).

In this chapter, we build upon existing work on query refinement for personalized information access [21] and start to extend it in the following ways:

- We describe a framework for information access that combines query refinement and relaxation in order to provide robust, personalized access to heterogeneous RDF data.
- We propose an implementation of the framework in terms of conditional rewriting rules for RDF query patterns.
- We discuss the application of the framework in the context of an existing e-learning system.

4.2 Background

The Background on robust querying is in the domain of open learning repositories where learning resources (courses, exercises, modules, etc.) are annotated with RDF metadata to allow users to find suitable material for his or her learning goal.

There are several characteristics of open learning repositories that are quite characteristic for RDF data in general and that make them a suitable text-case for our approach: Resources are authored by different people with different goals, background, domain expertise, etc. Providers of a resource can maintain the resource in proprietary databases. They might already have some personalization techniques implemented for the purposes of their specific context. They might employ user or learner models (which usually reflect applied techniques as well). User or learner features can already be maintained in human resource management systems, task management systems or user modeling servers. Furthermore, resources are accessed and consumed by people which differ in a wide range of characteristics. Learning in open environments demands effective personalization approaches to provide learner orientation and individualized access support.

4.2.1 Personalization by Query Refinement

In previous work, we have described a personalization service architecture for supporting users in finding learning resources in open learning environments. The central component of this architecture is the Personal Learning Assistant (PLA) Service [21] which integrates and uses other services to find learning resources, courses, or complete learning paths suitable for a user. The PLA accepts queries from a user and try to find the corresponding resources. The Personal Learning Assistant extends a user query by additional restrictions, joins, and variables based on various profiles. This extension is performed based on heuristic rules/functions. In the following, we briefly illustrate the kinds of heuristics used in the system based on an existing open learning repository for Computer Science Courses that contains about 2000 instances of learning resource taken from university courses.

```
SELECT * FROM
  {Resource} subject {Subject},
  {Resource} title {Title},
  {Resource} description {Description},
WHERE
  Subject Like "inference engines"
```

Figure 4.1: Basic Query

User queries to the open learning environment will consist of one or several keywords related to the topic the user wants to learn about. The result is a list of learning resources

including information about the subject and the title of the resource as well as a description of the content. In order to produce this list, the user request is translated into a query an RDF query language that matches the metadata use to describe learning resources in the system. Figure 4.1 shows the query corresponding to a user request for ”inference engines” in SeRQL syntax[1].

In a second step, the general query shown in figure 4.1 is adapted to better reflect the learning preferences of the user. In this step, the query is refined by extending it with additional constraints that are derived from the user profile. This is done by extending the path expression in the FROM and by adding variable assignments in the WHERE part of the query. Typical additions to a query are a restriction of the language of resources to the preferred language of the user and a general constraint demanding that the user must have all competencies that are required for understanding the resource.

```
SELECT * FROM
  {Resource} subject {Subject},
  {Resource} title {Title},
  {Resource} description {Description},
  {Resource} language {Language},
  {Resource} requires {} subject {Prerequisite},
  {User} hasPerformance {Performance},
  {Performance} learning_competency {Competence}
WHERE
  Subject Like "inference engines",
  Prerequisite = Competence,
  Language = de,
  User = user42
```

Figure 4.2: Query extended with user preferences

Figure 4.2 shows the result of refining the general query from figure 4.1 with language and competence constraints.

4.2.2 Problems with Refinement

In practice it turns out that the approach of personalization by query refinement suffers from serious problems. In fact problems occur in both steps of the query formulation process. The first problem already occurs when the basic query is formulated. In our open learning repository, this query does not return any result despite the fact that there are 8 resources on the subject. The reason for this is that only about 10% of all resources

[1]We omit namespaces for the sake of readability

are completely annotated with subject, title and description. Unfortunately, all 8 potential answers miss at least one of these properties and are therefore not returned as an answer. This problem can be reduced but *making predicates or triples optional in the query*.

Another problem lies in the fact, that the subject assigned to a course does not always correctly summarize the content. In our test data set for example, if the user provides the keyword ”Lernen” (German for ”learning”) no resources are returned despite the fact that there are resources for instance about Bayesian learning and learning in case based reasoning. The problem is here that in the case of the first resource the term learning only occurs in the title, but not the subject. In the case of the second resource, the term only occurs in the description and is mentioned neither in the subject nor the title of the resource. This problem can be used by *replacing triples/predicates for the others* based on domain knowledge.

We can observe the similar problems in connection with the refinement of the general query based on the user profile as the competence of a user is often defined in terms of learning resources that were successfully mastered by the student. This means that the subjects that represent the competency of a user are the subjects previously used resources and suffer the problems discussed above: If a resource lacks a subject, it cannot be added to the competencies. If the subject does not appropriately describe the content, the competencies of the user do not adequately reflect the actual state of knowledge etc. Therefore, a mechanism for *replacing predicate values* in query restrictions should be provided to solve the problem.

Another problem is connected for example with a request to match all prerequisite with user background knowledge. Sometimes it is enough, that a subset of user background knowledge for a resource is available in his profile to include particular resource in query results. Therefore, a mechanism to *replace quantifiers* of a query should be provided as well.

Further Problems arise from the inflexible nature of the rewriting mechanism that instantiates variables with the preferred value and leaves no room for taking the second best choice if the available resources are for example not in the preferred language, the user does not have all but most of the required competencies or the competencies of the user are not the same but very similar to the required ones. We will come back to these examples when we discuss our solution to the problem.

4.3 Rewriting RDF Queries

We propose an approach for query rewriting based on Event-Condition-Action (ECA) rules (see e.g. [80]) to solve the problem of over-constraint queries. This rewriting relaxes the over-constraint query based on rules and in order defined by events and conditions. This has an advantage that we start with the strongest possible query that is supposed to return the “best” answers satisfying most of the conditions. If the returned result set is

either empty or contains unsatisfactory results, the query is modified either by replacing or deleting parts of the query, or in other words relaxed. The relaxation should be a continuous step by step, (semi-)automatic process, to provide a user with possibility to interrupt further relaxations. Before we investigate concrete relaxation strategies in the context of our example domain, we first give a general definition of the framework for re-writing an RDF query.

Each resource is annotated with an RDF description which can be seen as a set of triples [44]. A query over these resources is formulated as triple patterns and a set of conditions that restrict the possible variables bindings in the patterns. Each triple pattern represents a set of triples. The corresponding abstract definition of a query focuses the essential features of queries over RDF; several concrete query languages are founded on these ideas including SeRQL which we use in our examples in figures 4.1 and 4.2.

Definition 4 (RDF Query) *Let $\mathcal{T}$ be a set of terms, $\mathcal{V}$ a set of variables, $\mathcal{RN}$ a set of relation names, and $\mathcal{PN}$ a set of predicate names. The set of possible triple patterns $\mathcal{TR}$ is defined as $\mathcal{TR} \subseteq 2^{(\mathcal{T}\cup\mathcal{V})\times(\mathcal{RN}\cup\mathcal{V})\times(\mathcal{T}\cup\mathcal{V})}$. A query Q is defined as the tuple $\langle TR_Q, P_Q\rangle$ with $TR_Q \in \mathcal{TR}$ and $P_Q \subseteq \mathcal{P}$ where $\mathcal{P}$ is the set of predicates with name $\mathcal{PN}$, defined over $\mathcal{T}$, and $\mathcal{V}$.*

The triple patterns TR_Q in a query Q determine those ground triples where a substitution τ exists. Formally a substitution τ is a list of pairs (X_i, T_i) where each pair tells which variable X_i has to be replaced by $T_i \in \mathcal{T} \cup \mathcal{V}$. Applied to a query, the substitution τ replaces variables in TR_Q with appropriate terms. If $\tau(TR_Q)$ is equal to some ground triples then the substitution is valid. All valid substitutions constitute answers to the query. The predicates P_Q restrict these substitutions additionally because only those bindings are valid answers where the predicates, i.e. $\tau(P_Q)$, are also satisfied. The predicates define additional constraints for the selection of appropriate triples. Using this abstract definition, the query in figure 4.1 would be represented as

$$\begin{aligned} TR_Q &= (\{(Resource, subject, Subject),\\ &\quad\ (Resource, title, Title)\\ &\quad\ (Resource, description, Description)\},\\ P_Q &= \{like(Subject, \text{“}inferenceengines\text{”})\} \end{aligned}$$

where $Resource, Subject, Title, Description \in \mathcal{V}$, as well as $subject, title, description$, $\text{“}inferenceengines\text{”} \in \mathcal{T}$ and $like \in \mathcal{PN}$. Alternatively, we could use variables as placeholders for the relations and assign the concrete relation names to them as conditions that use the equality predicate. The corresponding definition of the example query would be

$$\begin{aligned} TR_Q &= \{(Resource, R1, Subject), \\ &\quad (Resource, R2, Title) \\ &\quad (Resource, R3, Description)\}, \\ P_Q &= \{R1 = subject, R2 = title, R3 = description, \\ &\quad like(Subject, \text{“}inferenceengines\text{”})\} \end{aligned}$$

where $Resource, Subject, Title, Description, R1, R2, R3 \in \mathcal{V}$, $subject, title, description, \text{”}inferenceengines\text{”} \in \mathcal{T}$ and $like, = \in \mathcal{PN}$. This later representation can be seen as a normal form for queries that makes it easier to formulate re-writings in a general way. For sake of readability we will refer in the following to the original form instead of the normal form.

Based on the abstract definition of an RDF query, we can now define the notion of a rewriting rule and rewriting process as such. We define rewriting in terms of rewriting rules that take parts of a query, in particular triple patterns and conditions, as input and replace them by different elements.

Here we employ the principle of ECA-rules (event-condition-action rules) [18, 79] for continuous relaxation of user queries. A rewriting rule formally consists of three parts: a *pattern*, a *replacement* and some *conditions*. The pattern corresponds to the event, i.e. in our case an occurrence of particular triple patterns or predicates in a query. The replacement contains the terms which will substitute the matched pattern in a query; the replacement can be seen as the action in the ECA principle. Conditions constrain the rewriting and determine when particular rule can be fired because the rewriting rule can only be applied if the conditions are satisfied. These conditions can be used to define certain relaxation strategies. In particular, we will see later that conditions can be based on user preferences or background knowledge about the domain.

Definition 5 (Rewriting Rule) *A rewriting rule R is a 3-tuple $\langle PA, RE, CN \rangle$ where PA and RE are RDF queries according to Definition 1 and CN is a set of predicates.*

For conditions the same constructs as for queries are used where the possible results are also constrained by predicates. Patterns and replacements formally have the same structure like queries. They also consist of a set of triples and predicates. But patterns normally do not address complete queries but only a subpart of a query. Using this definition we can specify a rewriting rule that extends the simple query in figure 4.1 with the language preference of the user 42.

$$\begin{aligned} PA &= (\{(Resource, title, Subject)\}, \emptyset) \\ RE &= (\{(Resource, title, Subject), \\ &\quad (Resource, language, Language)\}, \\ &\quad \{Language = X\}) \\ CN &= \{languagePrefernce(User, X)\} \end{aligned}$$

where $languagePrefernce$ is a predicate which looks in his user profile for the language preference of $User$ who is in our case user 42.

While this example contained a rule for refining a query, we will see later that we can use the same mechanism for defining relaxations on a query.

In general a rewriting rules is applicable to all queries which contain the pattern at least as a part. The pattern does not have to cover the whole query. Normally it addresses some triples as well as some predicates in the query. In order to write more generic rewriting rules the pattern must be instantiated which is done by an substitution.

Definition 6 (Pattern Matching) *A pattern PA of a rewriting rule R is applicable to a query $Q = \langle TR_Q, P_Q \rangle$ if there exists two subsets $TR'_Q \subseteq TR_Q$ and $P'_Q \subseteq P_Q$ and a substitution θ with $\langle TR'_Q, P'_Q \rangle = \theta(PA)$.*

In contrast to term rewriting systems [5] the definition of a query as two sets of triples and predicates simplifies the pattern matching, i.e. the identification of the right subpart of the query for the pattern match. A subset of both sets has to be determined which must be syntactically equal to the instantiated pattern. Please note that due to set semantics, the triples and predicates in the pattern may be distributed over the query.

Now we will define how the new rewritten query is constructed with the help of the rewriting rule and pattern matching.

Definition 7 (Query Rewriting) *If a rewriting rule $R = \langle PA, RE, CN \rangle$*

- *is applicable to a query $Q = \langle TR_Q, P_Q \rangle$ with subsets $TR'_Q \subseteq TR_Q$ and $P'_Q \subseteq P_Q$ and substitution θ and*
- *$\theta(CN)$ is satisfied,*

then the rewritten query $Q^R = \langle TR^R_Q, P^R_Q \rangle$ can be constructed with $TR^R_Q = TR_Q \setminus TR'_Q \cup \theta(TR_{RE})$ and $P^R_Q = P_Q \setminus P'_Q \cup \theta(P_{RE})$ with $RE = \langle TR_{RE}, P_{RE} \rangle$.

Informally spoken, if the pattern match to a query and the conditions are satisfied then the matched pattern is substituted by the replacement. Applied the above rewriting rule for to the basic query we get the following refined rule:

Please note that the language preference of user 42 is “de” which means German.

4.4 Domain-Dependent Relaxation for Personalized Access

The formal apparatus introduced in the previous section provides us with a general mechanism for refining and relaxing RDF queries based on certain patterns and conditions. We have developed a specialized rule language that implements this mechanism which we will discuss in this section. In order to successfully use this language to relax over-constraint queries, we need a strategy for successively applying relaxations in such a way that we find answers that match the interests of the user as closely as possible and implement it in terms of query rewriting rules. The main problem with this approach and with query relaxation in general, is the fact that it is almost impossible to find generic relaxation strategies that work well across different applications. A good strategy rather depends on many factors including the nature of the information and the goals of the user. A solution for this problem is to employ explicit knowledge to drive the relaxation of a query. Corresponding to the factors that influence the usefulness of a strategy, there are two sources of knowledge we use for relaxation:

- Domain and Application knowledge;
- Knowledge about the user and user preferences.

The former represents a domain knowledge about dependencies between predicates and ordering according to their importance for queries within a domain. The second type of knowledge concerns the interests of the users and his profile. For example, to correctly determine the content of a resource in e-learning domain, we should first look into the subject, then the title, and finally the description in its metadata. This information can be used to constrain rewriting rules for the subject of the target resource in the title of the resource and the rule that looks for it in the description, in a way that the rewriting for title is performed before the rewriting for the description. Therefore, the order of importance between predicates in a domain serves as an order in which the rewriting rules should be applied. We will show later how this approach can be implemented using our rewriting approach. Another example relates to the structuring of the domain. If for example the language of a learning resource is not mentioned in the metadata, we can for example look at other learning resources that are part of the same course.

The second type of knowledge concerns the interests of the users. These interests are hard to determine automatically as they are influenced by many factors. A common approach is to use an explicit model of user preferences in terms of a user model. In the context of e-learning, this user model contains information about topics of user's interest, previous knowledge and preferences with respect to the type and format of learning resources, and so on.

As described in previous section, the rewriting rules are provided in terms of patterns (events), conditions, and replacements (actions). Figure 4.3 depicts a high level archi-

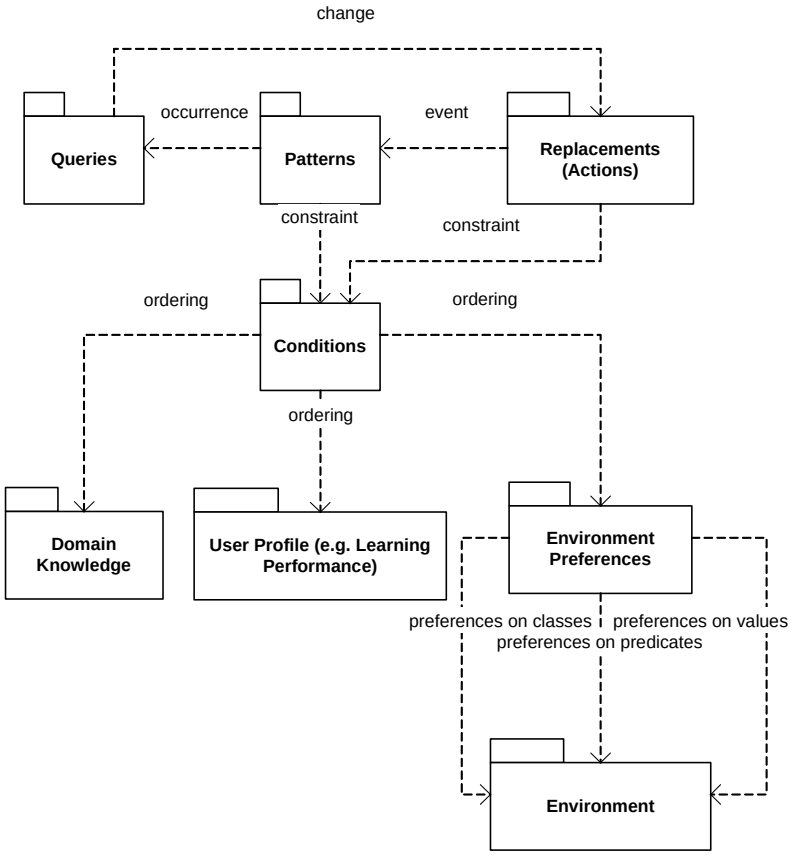


Figure 4.3: An architectural view on the components used within the rewriting process

tectural view on our rewriting system. We employed the Unified Modeling Language (UML) [39] package diagram. The boxes represent packages and the relations represent dependencies between the packages. We assume a generic application environment where presentation elements which are depicted at a user interface are characterized by an ontology described in RDF/RDFS. This includes for example fill in input boxes to type search terms, column descriptions in result sets, and so on. Each object of the user environment can be then described as an abstract environment concept which instantiates class, predicates, or values from domain ontology used in an application. This is represented as *Environment* package in the figure 4.3. Such an approach to represent environment allow us to express user preferences on the environment concepts by pointing to the environment concepts, predicates and values. This is represented by an *Environment Preferences* package and corresponding dependencies to the *Environment* package. User preferences can be ordered by an importance relation. This *ordering* together with the ordering derived from other parts of user profile (*User Profile* package) and ordering in *Domain Knowledge* is used to generate conditions determining the order of relaxation steps (*Conditions* package). The conditions constrain the query rewriting rules which consist of *Patterns*

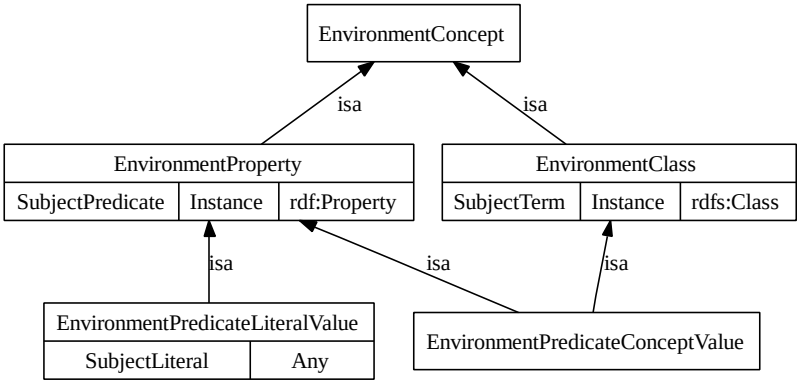


Figure 4.4: A Schema for Generic Environment

and *Replacements*. The set of generated queries is represented as a *Queries* package in figure 4.3. The *occurrence* of a pattern in a query is a triggering *event* for the *Replacements* actions. The set of *Queries* is dependent on the triggered *Replacements*; i.e. the replacements *change* the *Queries* package. Note that just a query which was produced as a last one is considered in each step of rewriting process. The details of the packages from the figure 4.3 are described in the following sections.

4.4.1 Environment and Preferences

In order to include knowledge about the domain of interest and the preferences of the user into the query relaxation process, we have designed a general scheme for representing relevant knowledge independent of a concrete application. This general scheme exploits the meta-modeling capabilities of RDF to define aspects of the world we can take into account in the rewriting process (compare fig. 4.4).

The schema follows an idea, that each environment can be generated according to an application domain schema used by the application. Rather than directly representing domain knowledge or user preferences it provides metaclasses that can be instantiated by existing representation schemes for information resources such as Learning Object Metadata (LOM) [75] as well as metadata schemas like the Dublin Core standard [23], and taxonomies and ontologies used for predicate values in the information resource schemas such as ACM computing classification system [77].

Environment concept can be for example linked to a field on a user interface form where the user can type a search term or it can be filled in with a class from a taxonomy. Such a generic environment schema provides us with a flexibility to describe any user environment which is based on schemas. For example, an environment concept can model a field on an entry form which is used to enter a subject term a user is searching for in the metadata. Such a field will be an instance of `EnvironmentProperty` class pointing

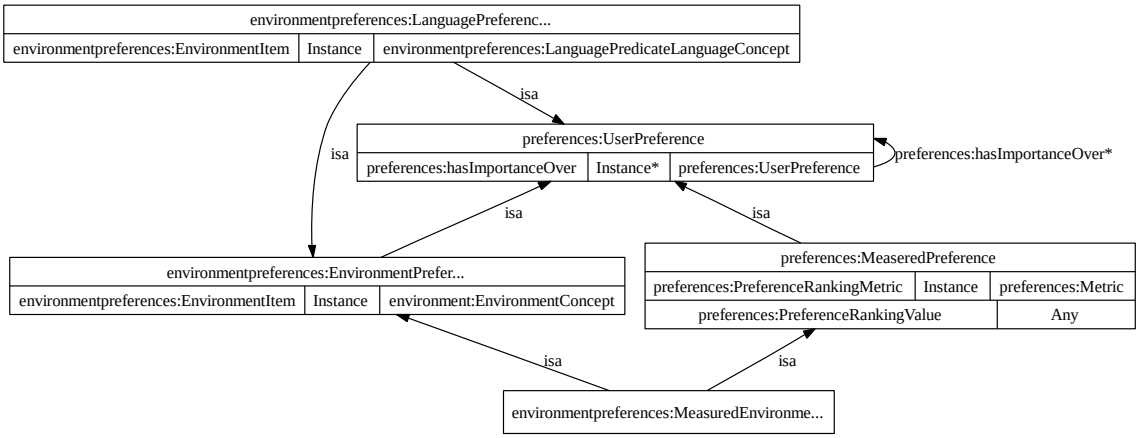


Figure 4.5: A Schema for Environment User Preferences

to a dc:subject predicate of the Dublin core schema. An example of combined class and predicate instance of an environment would be a predicate dc:subject with a class from a taxonomy like ACM CCS as its value.

Another advantage of such a generic environment schema is that we can refer to environment concepts from user preferences. Figure 4.5 depicts a schema for environment user preferences. Each user can express his level of preference for any environment concept. This is reflected by the `EnvironmentItem` property of the `EnvironmentUserPreference` class. Classes for environment preferences are further specialized according to which environment concept class is used to describe them. The level of a user preference can be expressed as a value from a metric. This is modeled by the `MesuredPreference` as a subclass of a user preference. The values from preference measures can be used to order them, i.e. to deduce the ordering relations between preference instances which is modeled by `hasImportanceOver` relation of user preference. Besides the user preferences, we also consider schema of a user background. This is represented as a learning performance and skills gained by performing learning. We use our schema for such a learner's learning performance [22] where the learning performance is described by a relation to learning competence, portfolios created and certificates gained during/from learning activities which have been connected to the learning performance.

To show a concrete instance of the environment preferences of a user, let us now consider a situation where a user John prefers a German language. In addition, he has attended two lectures, one on predicate logic and one on modal logic. An instance reflecting this situation described according to the environment user preference schemas is depicted in figure 4.6. John has a profile User1. His profile points to two performance objects: User1P1, and User1P2. The User1P1 is a performance record from the predicate logic lecture where user learned about inference engines (I.2.3.2 of ACM CCS) and backtracking (I.2.8.0 of ACM CCS). The User1P2 is a performance record from modal logic lecture where user learner about the modal logic concepts (I.2.4.1 of ACM CCS).

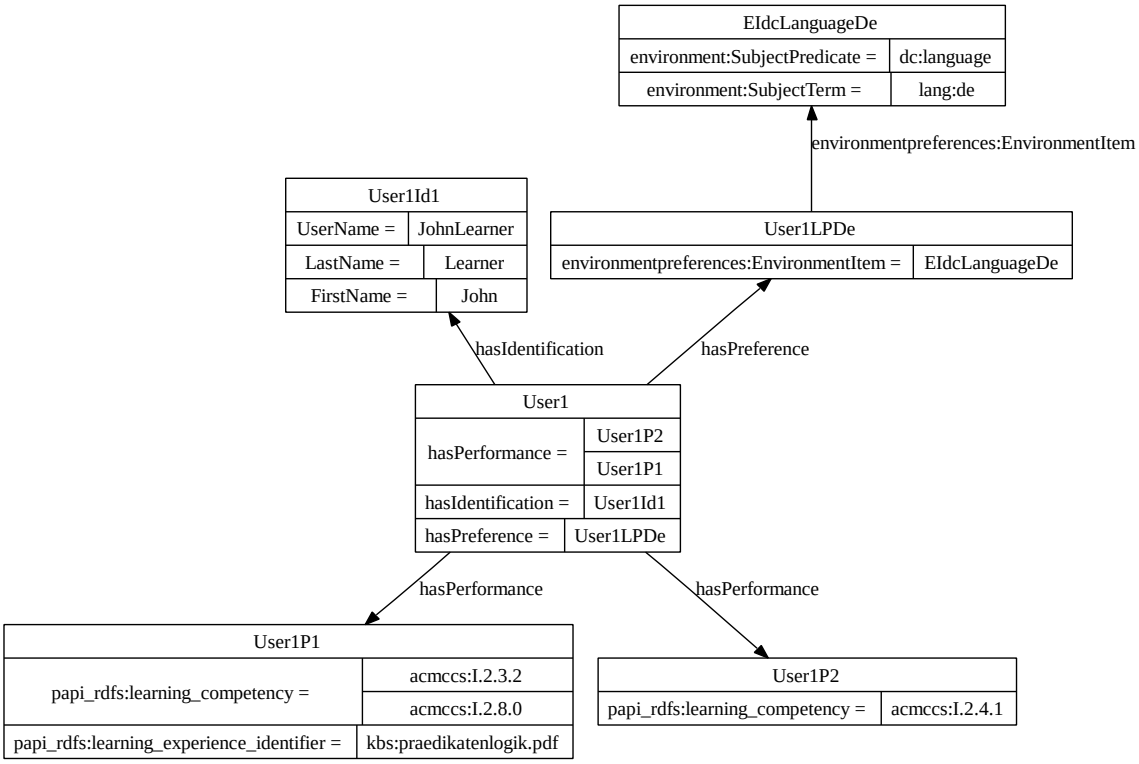


Figure 4.6: An Excerpt of Instance Examples for Environment User Preferences

The User1 profile also points to one preference object: User1LPDe. This is a language preference referring to a German language (lang:de).

This explicit representation of user preferences based on elements from the domain can be used to drive our rewriting process as it connects elements from the domain that occur in query expressions with user preferences. Using the hasImportanceOver relation, we can now decide which parts of an over-constrained query to relax first. By encoding this connection in RDF, we can use RDF queries to determine these preferences and specify the rewriting rules accordingly. In the following, we describe how preference knowledge about the domain and user can be used to relax over-constrained queries in our e-learning example.

4.4.2 Domain Knowledge and Relaxation

In general the rewriting is a very powerful approach in order to manipulate the overconstrained query. With replacing parts of a query we can realize five types of actions:

- *Making Triples/Predicates optional* — this provides a query which considers a situation that some of the triples/predicates do not have to appear in metadata. A query

then gives also results where particular predicate relaxed to an optional predicate does not occur;

- *Replacing Value* — this provides a query where particular predicate value is replaced with another value. Taxonomies may be used to provide siblings, more general terms, and so on;

- *Replacing quantifiers and operators* — this provides a query where quantifiers (like forall and exists) are replaced for each other. It also includes operators replacement like AND for OR, equal for a range, and so on;

- *Replacing Triples/Predicate* — this provides a query where particular triple resp. predicate in restrictions is replaced by another triple resp. predicate. A domain knowledge is employed for this purposes. For example, if a subject query is not satisfied, it may be replaced by title query with similarity measures;

- *Deleting Triple/Predicate* — this provides a query where particular predicate is deleted from a query completely.

As such, these operations are independent of the application domain and the user preference. A connection to the knowledge described above is made through the elements of the query that are affected by the corresponding operation. In most cases, we can identify a certain property that is affected. For example, in the learning environment a user searches a resource with a specific subject. But if there is no resource with that subject then we would like to relax the query that the subject term can also appear in the title of a resource description. This strategy can be derived from an environment preference stating that the ”subject” relation has the highest priority as it can be assumed to most precisely reflect the content of a resource followed by the ”title” and finally the ”description” relation. In the environment preference model, this order is described in terms of the hasImportanceOver relation. For the actual relaxation process each of these relations is implemented by a rewriting rule. The fist rewriting rule for that relaxation is specified in Figure 4.7.

Obviously, `PATTERN` defines the pattern, the `REPLACE-BY` the replacement, and `WITH` the conditions for the rewriting. The pattern contains one triple and one predicate. The triple {`Resource`} `subject` {`Subject`} looks for any resource `Resource` with subject `Subject`. The predicate `Subject Like Value ^^ xsd:string` constrains the variable `Subject` to the user's term (`Value` of type string), i.e. the subject the user is looking for. If a query contains such an triple and such an predicate then the rewriting rule is applicable.

The replacement part of the rule defines how the matched triple and predicate has to be replaced. The triple is extended by the second triple {`Resouce`} `title` {`Title`}. The second triple now allows to refer to the title of a resource. The first triple about the subject of the resource is not removed because there may be some other triples or

```
PATTERN
  {Resource} subject {Subject}
  WHERE
  Subject Like Valueˆˆxsd:string,
REPLACE-BY
  {Resource} subject {Subject},
  {Resource} title {TMPTitle}
  WHERE
  TMPTitle Like NEWTitle
WITH
  NEWTitle = concat("*",concat(Value,"*"))
```

Figure 4.7: Simple rewriting rule

predicates in the query which may refer to the subject of the resource. But the predicate is no longer needed and is completely substituted by the predicate `TMPTitle Like NEWTitle`, which now try to constrain the title of the resource instead of the subject. The variable `NEWTitle` is determined in the conditions. With the build-in function `concat` the value is prefixed and finished with a star which means that the title must only contain the subject the user is looking for.

The rewriting rule from figure 4.7 can be applied to the query in Figure 4.2. The result is shown in Figure 4.8. Note that now the query refer twice to the title of a resource. The second reference was introduced by the rewriting.

```
SELECT * FROM
  {Resource} subject {Subject},
  {Resource} title {TMPTitle},

  {Resource} title {Title},
  {Resource} description {Description},
  {Resource} language {Language},
  {Resource} requires {} subject {Prerequisite},
  {User} hasPerformance {} learning_competency {Competence}
WHERE
  TMPTitle Like "*inference engines*",
  Prerequisite = Competence,
  Language = de,
  User = user42
```

Figure 4.8: Relaxed query extended with user preferences

4.4.3 User Preferences and Relaxation

Another kind of relaxation is the rewriting the overconstrained query according to the knowledge about the user. In the learning scenario the user might prefer learning resources in German but Dutch may also be okay. This knowledge is used to refine the query, i.e. looking for resources in German. However if there is no resources in German then the query can be relaxed according to user's second preference.

As described above, our environment preference model allows user to specify an importance order between predicates. In contract to the domain preferences mentioned in the last section that can be specified inside the application, these preferences can be different for each user. As a consequence we have to provide an interface where each user can specify his or her personal preferences that can then be stored in the user profile (compare figure 4.6). A user interface for that is very simple, a slider is provided next to each item at a user interface for specifying an importance of the predicate for a user. The default slider positions are provided according to a default domain knowledge (compare figure 4.10). Using our general environment model, these preferences can be used in the same way as domain preferences once they have been entered by the user. In particular, the hasImportanceOver relation then defines conditions which are satisfied just when particular predicate is on its turn to rewrite it.

4.4.4 Conditions for User-constrained Relaxation.

Conditions play a crucial part for rewriting queries according to user's preferences. They can control when particular rewriting rule can applied.

Formally, conditions can be predicates where variables which are used in the pattern and the replacement are set together with some built-in functions for manipulating strings or numbers. The condition in the simple rewriting rule of Figure 4.7 is such an example; the used predicate is equality. But a condition can also be a query which should return at least one result in order to be satisfied and to bind variables to the values returned by the query. In this case, a query behaves like a normal predicate. But only the first result will be used during the rewriting; further results will be ignored. We use such queries to refer to users profile and user preferences.

An example is given in Figure 4.9. The rule try to relax user's first language preference to his second reference as stored in his profile. The condition of that rewriting rule starts from the root of his profile (`User`) to find his two preferences `Preference` and `Preference2` whereas the first preference is preferred to the second preference (relation `hasImportanceOver`). Both preferences refer to an environment item of type `EnvironmentPredicateConceptValue` with `subjectTerm language`. Their values are addressed in the pattern resp. replacement. The value of the first preference is replaced by the value of the second preference. The above rewriting rule can easily be generalized to a rewriting

```
PATTERN
  {Resource} language {Language}
  WHERE
  Language = L,
  User = UserID
REPLACE-BY
  {Resource} language {Language}
  WHERE
  Language = L2,
  User = UserID
WITH
  SELECT * FROM
   {User} hasPreference {Preference},
   [{User} hasPreference {Preference2}],
   {Preference} hasImportanceOver {Preference2},

   {Preference} EnvironmentItem {Item},
   {Item} type {EnvironmentPredicateConceptValue},
   {Item} subjectPredicate {language},
   {Item} subjectTerm {L},

   {Preference2} EnvironmentItem {Item2},
   {Item2} type {EnvironmentPredicateConceptValue},
   {Item2} subjectPredicate {language},
   {Item2} subjectTerm {L2}
  WHERE
   true
```

Figure 4.9: Rewriting rule for language preferences

for any value preference which the user related with the relation `hasImportanceOver`. The condition then would be that the environment items of such preferences must not refer to the subject predicate `language` but only to the same predicate (which is represented as a variable). So a variable instead of the literal `language` in the condition yields into a general rewriting rule for value preferences.

4.4.5 Ordering different rewriting rules

Conditions help to control the application of rewriting rules because they can implement a user-suggested ranking over the application of rewriting rules. For example, the rewriting rules will first replace a value of predicate dc:language originally comparing to “German” for value “Dutch” and then for value “English” for user preferring resources in German

before Dutch and Dutch before English.

However, there might be the problem that several rewriting rules are applicable to the same query. This situation might happen for example if the user specifies just preference order between values but does not specify an order between predicates. For example, he can give an order between the languages and formats of resources but between predicates for language or format. In that case, rewriting rules for relaxing the language and format requirements are applicable at the same which can cause overrelaxation.

A second problem which is related to the ordering is termination of relaxation. An interactive solution to these problems is letting the user decide which kind of relaxation he prefers in particular situation. The rewriting rules show him his possibilities. For example, [73] allows the user to select directions of relaxation and thus to indicate which relaxed answers may be of interest.

A naïve but automated strategy may be controlled by the number of returned answers. The number of results are counted globally; i.e. each relaxation step adds the number of results to the global counter. If the number of answers reaches a threshold, the relaxation is stopped. If any of the relaxation reaches the number of results greater then threshold further relaxations are not considered.

More promising approaches uses top-k [27, 40, 16] or skylining [64, 62]. Top-k needs a function which associates a ranking number with each answer. The k best answers then are returned. Obviously, the function should operate independently from the relaxation in order to select answers from different relaxations. Skylining assume several different dimensions. It tries to return the best answers according to each dimension. In this context the dimensions are the different possibilities of relaxations. Skylined relaxation returns the best answer from each possible relaxation.

4.5 Implementation Notes

We have implemented the rewriting approach as an extension of personalized search service of personal learning assistant [21]. The prototype combines user preference elicitation with user query formulation dialog. The original version of the personalized search had just a query formulation for restrictions of subject of resources. We have extended the user interface with generating environment based on the environment schema, a default environment for novice users, and a user preference elicitation. A user interface of such a personalization search environment is depicted in fig. 4.10.

The default environment consists of items for specifying subject concepts, title, description, and language as query literals. Each of the attributes on the user interface have a preference elicitation slider. The slider is used to specify a value measuring an importance of a preference of particular attribute to a user. These values are then used to derive an order in which the attributes will be processed in the query rewriting; i.e. the order of preference importance. In addition, user can specify value preferences for attributes

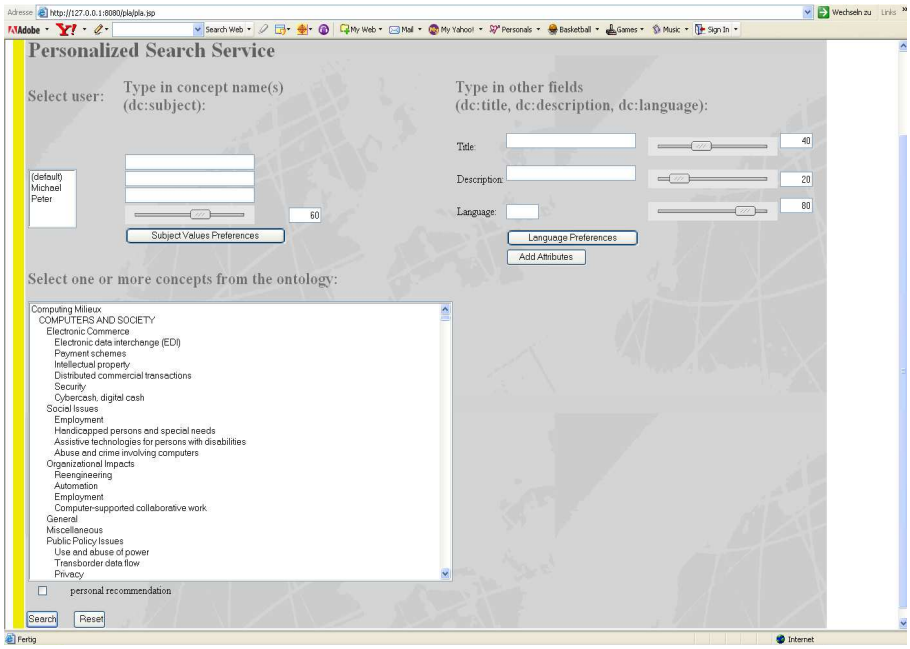


Figure 4.10: A user interface for personalized search based on user preferences

where a taxonomy of values is used. This is the case for example for language and subject preferences. A button for opening a dialog where a user can specify the value preferences and their order is provided where it is appropriate (e.g. *Subject Values Preferences* button or *Language Preferences* button). The source of values for subject preferences is in our case the ACM CCS taxonomy, used also for selecting concepts on the user dialogs. We use standard set of language identifiers as a source for values for language preferences. The value preference dialog displays a tree, a graph or a set of concepts with value labels determining the importance of the preference. When a user points to a concept, a slide bar is drilled down to change the preference importance value. If user needs to extend her restrictions, he can do that by selecting from other schema attributes which are offered when he presses *Add Attributes* button. The attributes which a user filled in on the user interface are used to construct the restriction part of queries.

The query results interface is organized into sub views. Each step in query rewriting results in separate html table with rows from the query result set. The most restricted query results are on top following step by step the queries created by query rewriting component.

The query results user interface is described using our environment ontology as well. The environment contains the attributes which should be displayed on the user interface for query results. These attributes are used in the projection part of the query (select part). The recommendation strategy on results is kept, i.e. a user gets a recommendation also based on the results according to her background knowledge as it was in the original

prototype. This is indicated by traffic light metaphor — recommended resources are green, may be recommended resources are yellow, and not recommended resources are red (see [21] for details).

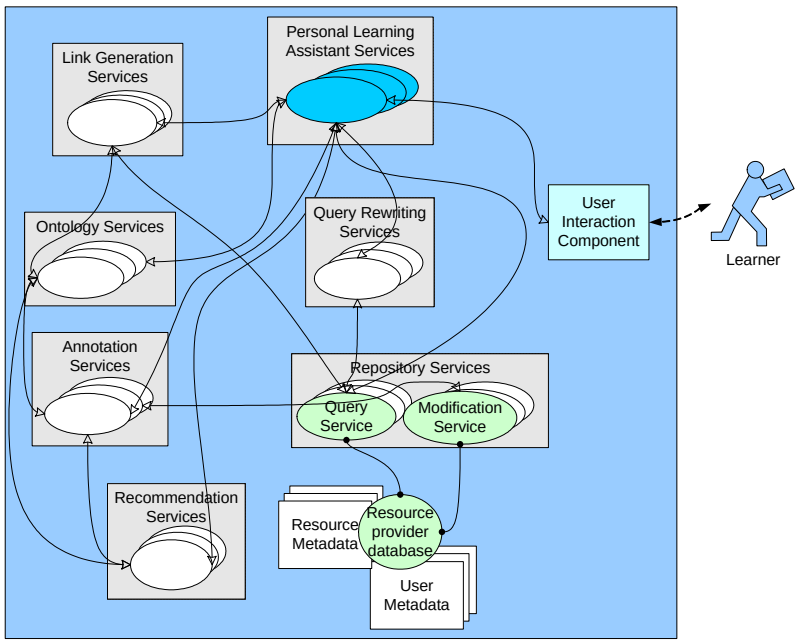


Figure 4.11: An architecture for personalized search based on query rewriting adopted from ([21])

Due to flexible architecture adopted from [21] (fig. 4.11), we have just replaced the components for query rewriting and repository access. The query rewriting component is written in prolog, which understands the rewriting strategies written in the language we have proposed here. In addition, the rewriting components take user preferences as an input. The preferences are used to dynamically build the conditions determining the order of the events to be fired; i.e. the order in which the rewriting rules are applied. We have used Sesame RDF database as a repository service. We have experimented with two metadata sets: the EU/IST Knowledge Web REASE (http://rease.semanticweb.org/ubp) and metadata from ULI project (http://www.uli-campus.de/). We have imported metadata from both repositories to the SESAME RDF database.

4.6 Related Work

Query relaxation have been studied in the context of cooperative query answering where information systems explicitly attempt to cooperate with their users [28]. Relaxation is a form of generalization where the scope of a query is extended through rewriting so that more information can be gathered in the answers. A query rewriting approach in deductive

databases with a help of specialized clauses is presented in [29]; if the head appears in a query then the head is replaced by the body of the specialized clause. A variant is presented in [30, 31] where users preferences are directly annotated to the logical atoms.

The work presented here shares the principle of query relaxation but proposes a framework for the semantic web meta data. As we described in Section 4.2, the semantic web environment differs from the databases in its less strict way of annotating the resources and heterogeneity. Therefore, more dimensions of knowledge about where to find the missing information have to be considered in a relaxation framework. Furthermore, our approach is more expressive due to the fact that the pattern of a rewriting rule can refer to more than one item [29] and use the more intuitive matching than unification.

Our approach also relates to term rewriting systems (see e.g. [5]) and graph transformation (see e.g. [82]) Our approach assumes the RDF queries where the set semantics of triple patterns and the conditions simplify considerably the rewriting process.

Furthermore, query relaxation approaches based on query rewriting (and many term rewriting approaches) proposed so far lack on conditionals. In our approach, we are able to define conditions which guard rewriting that it can take place just when the conditions are met. This allows us control the rewriting process. Moreover, with the help of conditions in the rewriting rules we are able to incorporate user profiles and user preferences which is separated from the data itself; annotating the RDF data of each distributed resource directly with all user preferences as proposed by [29] is not applicable for the semantic web.

Preference models have been studied in the fields of databases and artificial intelligence. A foundation on preference models in database systems has been given in [61]. A model for numerical and lexicographical preferences is given. An algebra which defines modification operators for such preferences is given as well. The preference model we have defined considers the partial order between several preferences similarly. We also allow for the numerical preferences ratings which is stored directly from a position of slider at the user interface. Contrary, we distinguish the predicate or schema preferences from the value preferences and their order. This allows us to order the relaxation steps in a way given by the order of preferences.

A query relaxation approach for discovering web services matching user goals has been proposed in [8]. They define preference model as a domain ontology, similarly to our approach. The approach differs in an algorithm for computing relaxation order. The main difference is that the preference model does not deal with ordering between predicates and values separately. It assumes them as bound together. Furthermore, it does not consider any ordering relations between the predicates. The order of relaxation is then computed according to combinations provided as levels. The levels are computed according to siblings in preference ontologies which are considered in several query predicates. As oppose, in our approach we consider order between the predicates and values separately. This gives us more informed strategy for computing order of relaxation steps.

Our environment and preference ontology relates to the work on CP-NETs [14] and

TCP-NETs [15] in artificial intelligence. The formalism allows to specify importance over variables and values as in our approach. This means that the model for environment preferences using semantic web formalism is transformable to the TCP-NETs. This allows to employ a reasoning about ceteris paribus used in preference models based on the TCP-NETs. In our approach, we have used the preference model for query approximation based event-condition-action and term rewriting system.

4.7 Conclusions and Further Work

In this chapter, we have proposed a framework for query relaxation to provide personalized information access to resources on the semantic web. The framework is based on the event-condition-action (ECA) paradigm where events are matching patterns, conditions are based on ordering between concepts of common sense domain knowledge and user preferences, and actions are the replacements for relaxing a query. The relaxation is based on the term rewriting principles enhanced with conditions provided by the ECA paradigm. This integration is a contribution to the term rewriting domain. The relaxation is controlled by conditions from domain and user preference ontology. The order is given by importance of predicates and values in the ontology for environment preferences, user profile, and common sense domain knowledge. This makes the approach very well suitable for the access to metadata on the semantic web as the domain knowledge helps to overcome the fact of heterogeneity and differences in how the metadata are authored on the semantic web.

In our further work, we would like to concentrate on ordering of the different rewriting possibilities and the algorithms for determining termination of relaxation. We have considered several strategies in this chapter but it requires further studies to give a recommendation how to decide among them. We also would like to experiment with different user preference models and how they contribute to the relaxation process. Last but not least, user preference elicitation methods and techniques needs to be studied to get as accurate user preferences as possible to support personalized access to information on the semantic web.

Chapter 5

DRAGO - Scalable Distributed Reasoning and Applications

by LUCIANO SERAFINI & ANDREI TAMILIN

In this chapter, we overview the theoretical and practical basis of distributed reasoning platform DRAGO (Distributed Reasoning Architecture for a Galaxy of Ontologies)[1] for dealing with multiple distributed ontologies interrelated by semantic links.

In particular:

- we start with a motivation of DRAGO for the semantic web and introduce its high level architectural vision;
- we recall major definitions of Distributed Description Logics framework (DDLs) [10, 87]. This framework forms a theoretical foundation for capturing the case of multiple ontologies interrelated by semantic links;
- we describe the intuitions and formal algorithms for performing distributed reasoning and simple distributed instance retrieval in DDLs;
- we describe the design and implementation principles of DRAGO reasoning platform which implements the distributed reasoning and querying algorithms for the case when ontologies are expressed in OWL [9] and interrelated by semantic links in C-OWL [11, 12];
- we give a simple working scenario describing the use of DRAGO in a step-by-step manner;
- we enumerate several applications of DRAGO for the semantic web;
- and finally we conclude and highlight the future directions to be taken.

[1] http://trinity.dit.unitn.it/drago

5.1 Motivation and Vision

Ontologies have been advocated as the basic tools to support interoperability between distributed applications and web services [2]. The basic idea is that different autonomously developed applications can meaningfully communicate by using a common repository of meaning, i.e., a shared ontology. The optimal solution obviously lies in having a unique worldwide shared ontology describing all possible domains. Unfortunately, this is non achievable in practice. The actual situation on the web is characterized by a proliferation of different ontologies. Each ontology describes a specific domain from different perspectives and at different level of granularity so that. This fact inevitably leads to a *heterogeneity* between ontologies describing even the very same domain. As a consequence, the initial problem of application interoperability passes to the level of ontology interoperability. Although the semantic standardization is far to be reached, the syntactic standardization is almost there, as it is widely accepted that ontologies should be expressed in OWL language, which is a variation of a descriptive language [9].

The common approach for enabling ontology interoperation is based on the definition of semantic relations between entities belonging to different ontologies, called a *semantic mapping*. A simple example of semantic mapping is the one stating that the concept Student in one ontology is more specific than the concept Person of another ontology. So far, several proposals of languages for expressing semantic mappings have been done. Some of them have a well-defined formal semantics, for example C-OWL [11], $\mathcal{E}$-connected OWL [36]. Examples of less formally grounded proposals are RDF Transformation [78] and MAFRA Semantic Bridge Ontology [69].

Given this situation, one of the challenges on the semantic web is of being able to deal with a large number of overlapping and heterogeneous *local ontologies*. We use the term "local" to stress the fact that each ontology describes a domain of interest from a local and subjective perspective. One of the most crucial aspects of ontology management on the semantic web is the capability of providing reasoning and querying services. Due to that fact, the problem of *reasoning* within and *querying* over a web of distributed, heterogeneous, and overlapping local ontologies interrelated by semantic mappings is of significant importance for enabling the practical semantic web.

Most of state of the art formalizations of that problem are based on the notion of a *global ontology* that allows to uniformly represent a set of local ontologies and semantic relations between them. In these approaches, reasoning in a set of local ontologies is rephrased into a problem of reasoning in the global ontology using state of the art reasoners [43, 51, 34].

The approaches based on the global ontology, however, present several main drawbacks. First, from a computational complexity point of view it is more convenient to keep the reasoning as much local as possible, exploiting the structure provided by semantic relations for the propagation of reasoning through the local ontologies. Some intuition in this direction can be found in the computational complexity results for satisfiability in Multi-Context Systems described in [88]. Second, the reasoning procedure that has to be

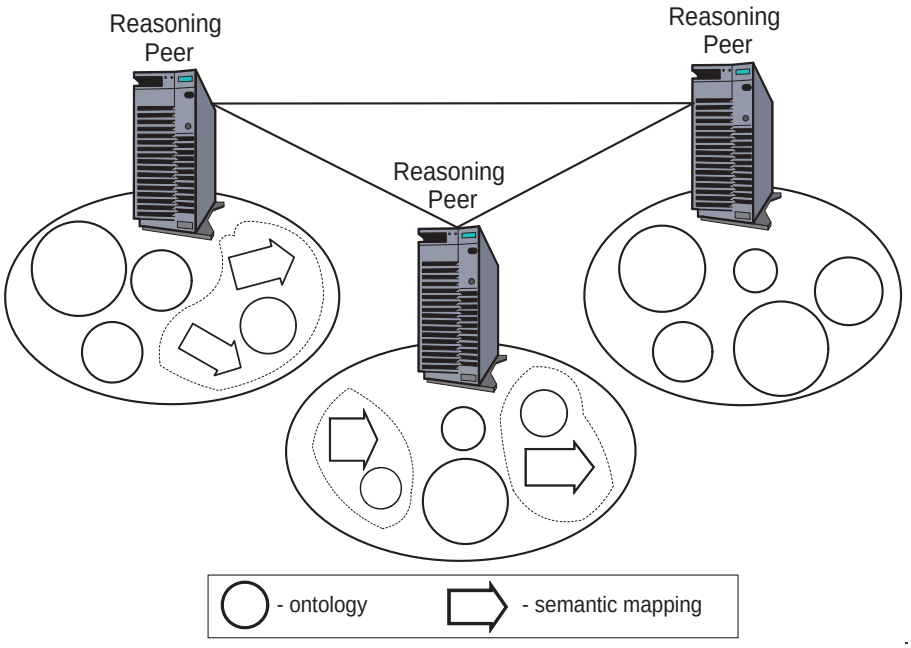


Figure 5.1: DRAGO reasoning architecture for the semantic web

implemented in the global ontology should be capable of dealing with the most general local language, whereas having a some distributed approach would allow to apply to every local ontology the specific local reasoner, optimized for the local language.

To overcome the pointed out disadvantages of global reasoning approach we pursue an alternative technique, which is based on the *distributed contextual reasoning paradigm*. Namely, the reasoning with multiple ontologies is proposed to be accomplished through a suitable combination, via semantic mappings, of local reasoning chunks, internally executed in each distinct ontology.

For enabling the distributed reasoning we propose an architecture called DRAGO (Distributed Reasoning Architecture for a Galaxy of Ontologies). From the architectural point of view, our vision is inspired by general peer-to-peer paradigm and particular distributed knowledge management architectures, such as the ones proposed in SWAP [24] and Edamok [71] projects, and by C-OWL language [11].

As depicted on Figure 5.1, DRAGO envisages a web of ontologies being distributed amongst a peer-to-peer network of *DRAGO Reasoning Peers* (hereafter, DRP). The role of a DRP is to provide reasoning and querying services for ontologies registered to it, as well as to request reasoning and querying services of other semantically related peers. The key difference of DRP from standard ontology reasoners (non distributed) is that it registers not just a stand alone ontology, but an ontology coupled with a set of semantic mappings. The reasoning and querying services provided by a DRAGO peer can be subdivided into two groups: (1) *local*, when semantic mappings are ignored, and (2) *distributed*, when behind a local ontology the semantically related ontologies are considered.

5.2 Distributed Description Logics Framework

Distributed Description Logics (DDLs) is a framework designed to formalize the case of multiple ontologies interconnected by semantic mappings. In this section, we recall the definitions of DDLs as given in [10, 87].

5.2.1 Syntax and Semantics of DDLs

Given a set I of indeces enumerating ontologies, a DDL is a collection of description logics $\{\mathcal{D}L_i\}_{i\in I}$ corresponding to each ontology. We will denote a T-box of $\mathcal{D}L_i$ as $\mathcal{T}_i$ and an A-box as $\mathcal{A}_i$. The semantic mapping between ontologies are expressed as "bridge rules" between pairs of connects belonging to different ontologies.

A bridge rule from i to j is an expression of the following two forms:

$i : x \xrightarrow{\sqsubseteq} j : y$ – an *into-bridge rule*

$i : x \xrightarrow{\sqsupseteq} j : y$ – an *onto-bridge rule*

where x, y are concepts or individuals belonging to $\mathcal{D}L_i$ and $\mathcal{D}L_j$ respectively. The derived bridge rule $i : x \xrightarrow{\equiv} j : y$ can be defined as a conjunction of the corresponding into- and onto-bridge rule.

Despite this general form allowing for unrestricted mix of concepts with individuals, e.g., nominals (classes with a singleton extension), in this chapter, we restrict the bridge rules to be expressions connecting only pairs of concepts. Therefore, bridge rules are allowed to connect only terminological parts of two DLs (ontologies). Moreover, we will prohibit the use of nominals in component logics $\mathcal{D}L_i$ of DDLs. This restriction is motivated by the hardness of dealing with nominals even in case of standard Description Logics [98, 99].

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

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

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

In order to express correspondences between semantically related heterogeneous individuals between A-boxes, we follow the approach of introducing the individual-level equivalent of bridge rules:

$i : x \mapsto j : y$ – an *individual correspondence*

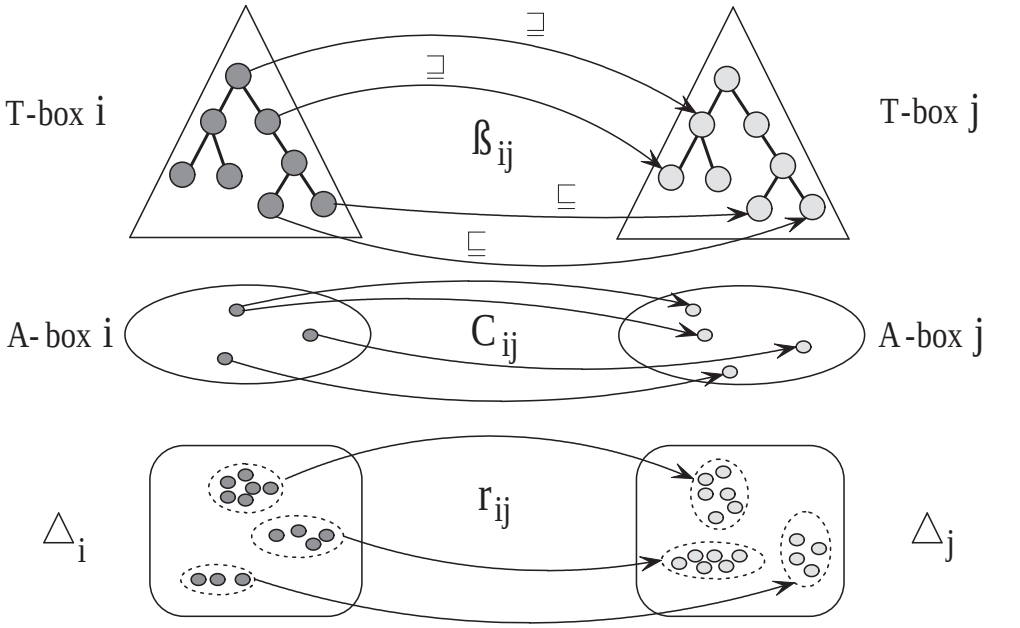


Figure 5.2: Graphical intuition of Distributed Description Logics framework

where x, y are individuals of $\mathcal{A}_i$ and $\mathcal{A}_j$ respectively.

Similarly to bridge rules, individual correspondences from i to j reflect a subjective point of view of the j-th ontology. For example, the correspondence $i : a \mapsto j : b$ intuitively expresses that according to j-th point of view, the individual b is one of the possible translations of the foreign individual a in the local domain of j. Notice, that this definition admits multiple translations of foreign individuals, i.e., one can simultaneously have $i : a \mapsto j : b$ and $i : a \mapsto j : b'$.

A *distributed A-box (DAbox)* $\mathfrak{A} = \langle \{\mathcal{A}_i\}_{i \in I}, \mathfrak{C} \rangle$ consists of a collection of A-boxes $\{\mathcal{A}_i\}_{i \in I}$ and a collection of individual correspondences $\mathfrak{C} = \{\mathfrak{C}_{ij}\}_{i \neq j \in I}$ between pairs of corresponding A-boxes.

A *distributed knowledge base (DKB)* $\mathfrak{K} = \langle \mathfrak{T}, \mathfrak{A} \rangle$ is then a pair containing a distributed T-box and a distributed A-box.

The semantics of DDLs, graphically depicted on Figure 5.2, locally interprets each ontology by a standard Description Logics interpretation on its local domain. Since local domains Δ_i may be heterogeneous, the semantic correspondences between heterogeneous domains are modeled using so called domain relation r_{ij} from Δ_i to Δ_j and defined as a subset of $\Delta_i \times \Delta_j$. For instance, if Δ_1 and Δ_2 are the representation of time as Rationals and as Naturals, r_{12} could be the round off function, or some other approximation relation.

A *distributed interpretation* $\mathfrak{I} = \langle \{\mathcal{I}_i\}_{i \in I}, \{r_{ij}\}_{i \neq j \in I} \rangle$ of a DKB $\mathfrak{K} = \langle \mathfrak{T}, \mathfrak{A} \rangle$ a consists of local interpretations $\mathcal{I}_i$ for each $\mathcal{D}L_i$ on local domains $\Delta^{\mathcal{I}_i}$ and a family of domain relations r_{ij} between these local domains.

To deal with possible inconsistencies, DDLs utilizes the notion of a special non-classical interpretation $\mathcal{I}^\epsilon$, called a hole (see [87] for details), that interprets every concept or role in the empty set. The hole satisfies any statement of knowledge base, even classically unsatisfiable (inconsistent). To highlight that distributed interpretation $\mathfrak{I}$ can contain holes, we will subscript the inferences w.r.t. $\mathfrak{I}$ with a symbol "ϵ".

A distributed interpretation $\mathfrak{I}$ is said to satisfy (written $\mathfrak{I} \models_\epsilon$) the elements of distributed T-box $\mathfrak{T}$ if

- $\mathcal{I}_i \models_\epsilon A \sqsubseteq B$ for all $A \sqsubseteq B$ in $\mathcal{T}_i$
- $\mathfrak{I} \models_\epsilon i : A \xrightarrow{\sqsupseteq} j : G$, if $r_{ij}(A^{\mathcal{I}_i}) \supseteq G^{\mathcal{I}_j}$ (A, G are concepts of $\mathcal{T}_i, \mathcal{T}_j$)
- $\mathfrak{I} \models_\epsilon i : B \xrightarrow{\sqsubseteq} j : H$, if $r_{ij}(B^{\mathcal{I}_i}) \subseteq H^{\mathcal{I}_j}$ (B, H are concepts of $\mathcal{T}_i, \mathcal{T}_j$)
- $\mathfrak{I} \models_\epsilon \mathfrak{T}$, if for every $i, j \in I$, $\mathfrak{I} \models_\epsilon \mathcal{T}_i$ and $\mathfrak{I} \models_\epsilon \mathfrak{B}_{ij}$

Finally, $\mathfrak{T} \models_\epsilon i : C \sqsubseteq D$ if for every $\mathfrak{I}$, $\mathfrak{I} \models_\epsilon \mathfrak{T}$ implies $\mathfrak{I} \models_\epsilon i : C \sqsubseteq D$. We say that $\mathfrak{T}$ is *satisfiable* if there exists a $\mathfrak{I}$ such that $\mathfrak{I} \models_\epsilon \mathfrak{T}$. Concept $i : C$ is *satisfiable* with respect to $\mathfrak{T}$ if there is a $\mathfrak{I}$ such that $\mathfrak{I} \models_\epsilon \mathfrak{T}$ and $C^{\mathcal{I}_i} \neq \emptyset$.

Concerning the assertional part, a distributed interpretation $\mathfrak{I}$ is said to satisfy the elements of distributed A-box $\mathfrak{A}$ if

- $\mathcal{I}_i \models_\epsilon C(a), \mathcal{I}_i \models_\epsilon R(a, b)$ for all assertions $C(a)$, $R(a, b)$ in $\mathcal{A}_i$
- $\mathfrak{I} \models_\epsilon i : a \mapsto j : b$, if $\left\langle a^{\mathcal{I}_i}, b^{\mathcal{I}_j} \right\rangle \in r_{ij}$ (a, b are individuals of $\mathcal{A}_i, \mathcal{A}_j$)
- $\mathfrak{I} \models_\epsilon \mathfrak{A}$, if for every $i, j \in I$, $\mathfrak{I} \models_\epsilon \mathcal{A}_i$ and $\mathfrak{I} \models_\epsilon \mathfrak{C}_{ij}$

Finally, $\mathfrak{A} \models_\epsilon i : C(a)$ ($\mathfrak{A} \models_\epsilon i : R(a, b)$) if for every $\mathfrak{I}$, $\mathfrak{I} \models_\epsilon \mathfrak{A}$ implies $\mathfrak{I} \models_\epsilon i : C(a)$ ($\mathfrak{I} \models_\epsilon i : R(a, b)$). We say that $\mathfrak{A}$ is *consistent* if there exists a $\mathfrak{I}$ such that $\mathfrak{I} \models_\epsilon \mathfrak{A}$.

A statement α is entailed by the distributed knowledge base $\mathfrak{K} = \langle \mathfrak{T}, \mathfrak{A} \rangle$ (written $\mathfrak{K} \models_\epsilon$) iff for every distributed interpretation $\mathfrak{I}$, if $\mathfrak{I} \models_\epsilon \mathfrak{T}$ and $\mathfrak{I} \models_\epsilon \mathfrak{A}$, then $\mathfrak{K} \models_\epsilon \alpha$.

5.2.2 Properties of DDLs

In the following, we intuitively recall paradigmatic properties of DDLs that are desirable for multi-ontology environments. For the formal description of properties we refer an interested reader to [10, 87].

Knowledge propagation Bridge rules from a source ontology i to a target ontology j constitute a *semantic channel* from i to j which allows ontology j to access and import knowledge from ontology i. In particular, bridge rules *propagate* across

ontologies a terminological knowledge in form of *subsumption axioms*. Also, a combination of bridge rules with individual correspondences propagate across ontologies an assertional knowledge in form of *concept membership assertions*.

Directionality Bridge rules from i to j support knowledge propagation *only* in such a direction from i towards j.

Isolation If there are no bridge rules that go from i towards j, then j is not affected by i. This says that an ontology without incoming bridge rules is not affected by other ontologies. Vice versa, an ontology without outgoing bridge rules does not affect the other ontologies.

Localized inconsistency The inconsistency of one ontology, or some subgroup of connected ontologies, does not automatically render the *entire* multi-ontology environment inconsistent.

5.3 Distributed Reasoning in DDLs

Reasoning is the fundamental process of discovering facts entailed by knowledge base. Although both in DLs and DDLs the fundamental reasoning task lays in a verification of concepts subsumption, in DDLs, besides the ontology itself, the subsumption depends also on other ontologies that affect it through the semantic mappings. In this section, we describe a decision procedure that computes DDLs subsumption and a distributed tableau reasoning algorithm for determining whether $\mathfrak{T} \models_\epsilon i : A \sqsubseteq B$.

5.3.1 Subsumption Propagation Mechanism

Before turning to the description of decision procedure, let us first recall a DDLs subsumption propagation mechanism that constitutes the main reasoning step in DDLs. The basic idea is that a combination of onto- and into-bridge rules allows for directional propagating the terminological knowledge across ontologies in form of DL subsumption axioms.

Formally this reasoning pattern is formulated according to the following proposition: Given a distributed T-box $\mathfrak{T} = \langle \{\mathcal{T}_i\}_{i \in I}, \{\mathfrak{B}_{ij}\}_{i \neq j \in I} \rangle$, if $\mathfrak{B}_{ij}$ contains $i : A \xrightarrow{\sqsupseteq} j : G$ and $i : B_k \xrightarrow{\sqsubseteq} j : H_k$ for $1 \leq k \leq n$ and $n \geq 0$, then:

$$\mathfrak{T} \models_\epsilon i : A \sqsubseteq \bigsqcup_{k=1}^{n} B_k \Longrightarrow \mathfrak{T} \models_\epsilon j : G \sqsubseteq \bigsqcup_{k=1}^{n} H_k \tag{5.1}$$

where $\bigsqcup_{k=1}^{0} D_k$ denotes $\bot$. Figure 5.3 depicts this result for the case of simple distributed T-box $\mathfrak{T}_{12} = \langle \{\mathcal{T}_1, \mathcal{T}_2\}, \mathfrak{B}_{12} \rangle$ when a single pair of into-onto-bridge rules specified.

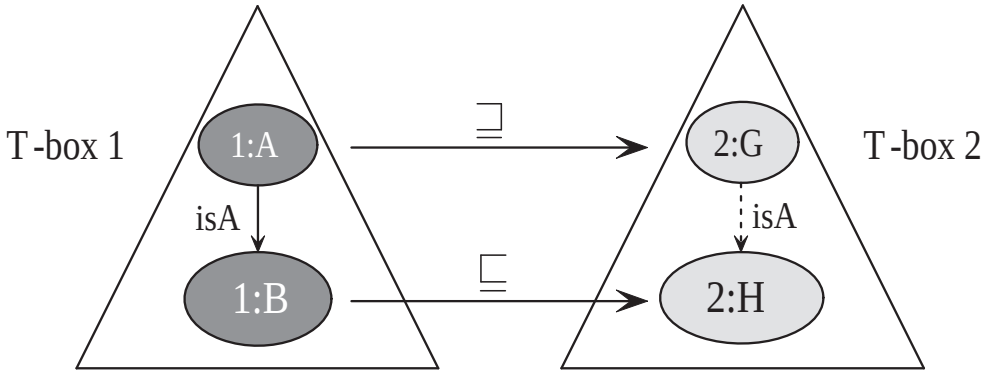


Figure 5.3: Example of subsumption propagation in DDLs (inferred subsumption is dashed)

Taking this reasoning pattern allows to define a *bridge operator* which encapsulates the subsumption axioms that has propagated via bridge rules.

Given a set of bridge rules $\mathfrak{B}_{12}$ from $\mathcal{D}L_1$ to $\mathcal{D}L_2$, the *bridge operator* $\mathfrak{B}_{12}(\cdot)$, taking as input a T-box in $\mathcal{D}L_1$ and producing a T-box in $\mathcal{D}L_2$, is defined as follows:

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

Notationally, $\bigsqcup_{k=1}^{0} D_k$ expresses $\bot$. It is also remarkable that these are essentially *all* the inferences that one can get according to the semantics of DDLs with holes. Thus, given a distributed T-box $\mathfrak{T}_{12} = \langle \mathcal{T}_1, \mathcal{T}_2, \mathfrak{B}_{12} \rangle$ we have that $\mathfrak{T}_{12} \models_\epsilon 2 : X \sqsubseteq Y$ if and only if $\mathcal{T}_2 \cup \mathfrak{B}_{12}(\mathcal{T}_1) \models X \sqsubseteq Y$.

For arbitrary family $\mathfrak{B} = \{\mathfrak{B}_{ij}\}_{i,j \in I}$ of bridge rules, we can then compose a combined new operator $\mathfrak{B}$ on a family of T-boxes as follows:

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

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

Finalizing description of subsumption propagation, we can state the correctness and completeness of the defined operator. Formally, for any $\mathfrak{T} = \langle \mathbf{T}, \mathfrak{B} \rangle$, $\mathfrak{T} \models_\epsilon j : X \sqsubseteq Y$ if and only if the j-th T-box of $\mathfrak{B}^*(\mathbf{T})$ entails $X \sqsubseteq Y$.

5.3.2 Distributed Tableaux Algorithm for DDLs

To simplify the description, we suppose that local ontologies are expressed in (a subset of) the $\mathcal{SHIQ}$ language — one of the most widely known DLs. Also, we will assume that the consequences of bridge rules are atomic names. This last condition can easily be achieved by introducing, through definitions, names for the consequent concepts. We need the usual notion of axiom internalization, as in [53]: given a T-box $\mathcal{T}_i$, the concept $C_{\mathcal{T}_i}$ is defined as $C_{\mathcal{T}_i} = \bigsqcap_{E \sqsubseteq D \in \mathcal{T}_i} \neg E \sqcup D$; also, the role hierarchy $R_{\mathcal{T}_i}$ contains the role axioms of $\mathcal{T}_i$, plus additional axioms $P \sqsubseteq U$, for each role P of $\mathcal{T}_i$, with U some fresh role.

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

For each $j \in I$, the function $\mathbf{DTab}_j$ takes as input a concept expression X and tries to build a representation of $\mathcal{I}_j$ with $X^{\mathcal{I}_j} \neq \emptyset$ (called a *completion tree* [53]) for the concept $X \sqcap C_{\mathcal{T}_j} \sqcap \forall U.C_{\mathcal{T}_j}$, using the $\mathcal{SHIQ}$ expansion rules, w.r.t. the role hierarchy $R_{\mathcal{T}_j}$, plus the following additional "bridge" expansion rules:

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

if 1. $G \in \mathcal{L}(x)$, $i : A \xrightarrow{\sqsupseteq} j : G \in \mathfrak{B}_{ij}$, and
2. *IsSat*$_i(A \sqcap \neg \bigsqcup \mathbf{B}') =$ *False*, for some $\mathbf{H}' \not\subseteq \mathcal{L}(x)$,

then $\mathcal{L}(x) \longrightarrow \mathcal{L}(x) \cup \{\bigsqcup \mathbf{H}'\}$

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

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

The idea, inspired by bridge operator $\mathfrak{B}_{ij}(\cdot)$, is that whenever $\mathbf{DTab}_j$ encounters a node x that contains a label G which is a consequence of an onto-bridge rule, then if $G \sqsubseteq \sqcup\mathbf{H}$ is entailed by the bridge rules, the label $\bigsqcup \mathbf{H}$, is added to x. To determine if $G \sqsubseteq \sqcup\mathbf{H}$ is entailed by bridge rules $\mathfrak{B}_{ij}$, $\mathbf{DTab}_j$ invokes $\mathbf{DTab}_i$ on the satisfiability of the concept $A \sqcap \neg(\sqcup\mathbf{B})$. $\mathbf{DTab}_i$ will build (independently from $\mathbf{DTab}_j$) an interpretation $\mathcal{I}_i$, as illustrated in Figure 5.4. To avoid redundant calls, $\mathbf{DTab}_j$ caches the calls to $\mathbf{DTab}_i$ in a data structure *IsSat*$_i$, which caches the subsumption propagations that have been

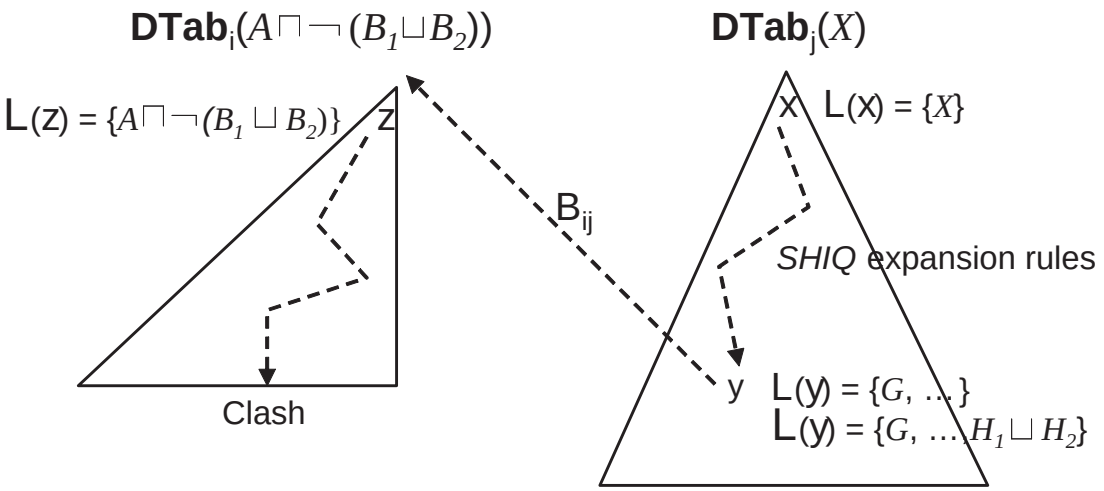


Figure 5.4: Illustrative step of the distributed tableaux algorithm: subsumption propagation forced by bridge rules $i : A \xrightarrow{\sqsupseteq} j : G$, $i : B_1 \xrightarrow{\sqsubseteq} j : H_1$ and $i : B_2 \xrightarrow{\sqsubseteq} j : H_2$

computed so far. Specifically, for every C, *IsSat*$_i(C)$ will be set to *True/False* whenever $\mathfrak{T} \not\models_\epsilon i : C \sqsubseteq \bot$ is determined.

Note, that the construction of the distributed interpretation can be parallelized, as each local tableau procedure can run independently from the others, without checking for standard blocking conditions with nodes generated by the other local tableau.

The proposed distributed tableaux algorithm can be easily generalized for the case of arbitrary distributed T-boxes. To prevent infinite looping due to the cycles in the bridge graph, the satisfiability requests are named with an id. Later on, the same id is used for naming all initiated satisfiability subqueries. Doing so allows to distinguish that a certain satisfiability request $j : X_1$ is a consequence of initial satisfiability request $j : X$ and thus needs to be properly handled, e.g., blocked. This later means that a satisfiability request should never generate itself.

What does it mean that a certain satisfiability request "returned back" through the cyclical bridge rules? – It means that the request couldn't be closed locally in none of the local tableaux and bridge rules do not propagate any contradiction.

Every tableaux procedure **DTab**$_j$, therefore, is extended to take as input parameters a concept expression X and an identifier id_X, i.e., **DTab**$_j(X, id_X)$. Before applying tableaux rules described above, **DTab**$_j$ checks whether X has been already asked and if so returns immediately *Satisfiable*. Otherwise the tableaux rules are applied and whenever necessary **DTab**$_i(\ldots, id_X)$ is invoked with the same identifier.

It worth stressing out that a localized inconsistency property of DDLs (see Section 5.2.2) is automatically reflected by the specified distributed tableaux. Indeed, the distributed tableaux procedure, say **DTab**$_j$, takes into account only those **DTab**$_i$ that return

Unsatisfiable, whereas a reasoner for an inconsistent ontology will always reply *Satisfiable* and thus will have no effect.

5.4 Distributed Query Answering in DDLs

The query answering is a second crucial requirement to ontology technology on the semantic web. One of the simple ontological queries defined in standard Description Logics is an *instance retrieval*, i.e., finding all instances of a particular concept. Intuitively for the case of multi-ontology environments, to guarantee the completeness of instance retrieval, besides locally retrieved instances one should also retrieve instances of other semantically related ontologies.

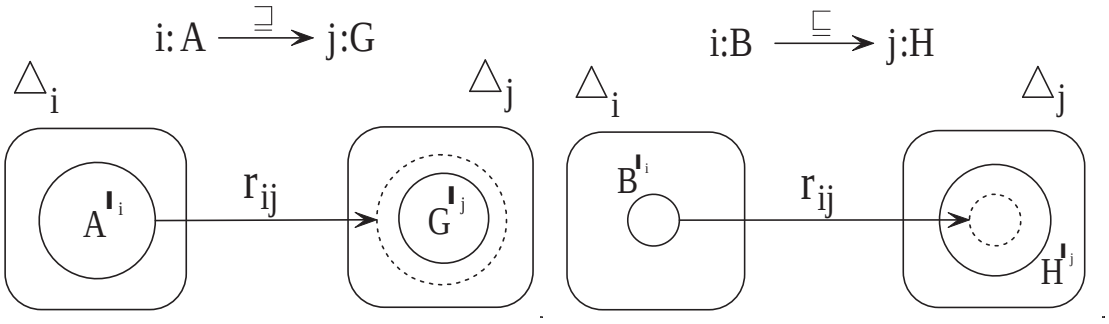
To clarify the idea of distributed instance retrieval, let us consider a simple scenario. The human resource offices of a university and one of its department utilize ontologies for describing research staff. The university exploits ontology O^{Uni}, while the department uses O^{Dept}. The obvious overlap of the domains of interest in both ontologies occurs since every person that works in the department belongs to the university. However, let us additionally suppose that O^{Uni} instantiates its concept $PersonContact$ in the domain of personal e-mails, while O^{Dept}, respectively, instantiates its concept $PersonContact$ in the domain of world wide web personal homepages.

When the ontology of the university is asked for all personal contacts, to guarantee a complete answer, this query should be propagated to the departmental ontology expanding the result set. However, since instantiations of semantically related concepts are expressed using different "formats", one needs a mechanism of semantic preserving instance transformation. In our example, when retrieving instances of $PersonContact$ in O^{Uni}, i.e., contacts of the university staff, besides locally found e-mails one should also augment the response by retrieving the instances of $PersonContact$ in O^{Dept}, i.e., personal homepages, and further project them into domain of O^{Dept}, i.e., "visit" every homepage found and extract, if any, personal e-mails from the pages.

In this section, we analyze the semantics of bridge rules and individual correspondences in order to understand their operational behavior w.r.t. propagating the assertional knowledge. Later on, this will allow us to define the distributed instance retrieval algorithm.

5.4.1 Assertional Propagation Mechanism

As we could see in Section 5.3, bridge rules constitute the channel for propagating across ontologies the knowledge in form of subsumption axioms. In order to get the intuition of the affect Let us now investigate the consequence of bridge rules and individual correspondences in order to understand their affect on assertional part of knowledge in distributed knowledge base. We visualize on Figure 5.5 the semantics given to bridge rules in DDLs:



(a) Onto-bridge rule (b) Into-bridge ruleÂ

Figure 5.5: Visualized semantics of bridge rules in DDLs

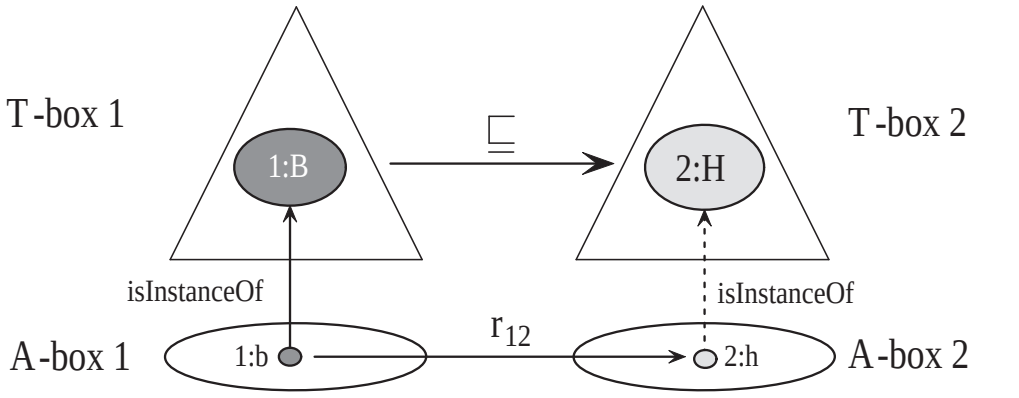


Figure 5.6: Concept membership propagation in DDLs (inferred assertion is dashed)

The special role of into-bridge rules can be intuitively grasped from these visualizations. Indeed, the semantics of into-bridge rule depicted on Figure 5.5(b) says: if in ontology i a concept B has an instance b then in ontology j there should exist such instance h of H that a pair $\left\langle b^{\mathcal{I}_i}, h^{\mathcal{I}_j} \right\rangle$ belongs to the domain relation r_{ij}.

Formally this intuition can be stated as an assertional propagation pattern: Given a distributed knowledge base $\mathfrak{K} = \langle \mathfrak{T}, \mathfrak{A} \rangle$, if $\mathfrak{B}_{ij}$ contains $i : B \xrightarrow{\sqsubseteq} j : H$ and $\mathfrak{C}_{ij}$ contains $i : b \mapsto j : h$, then:

$$\mathfrak{K} \models_\epsilon i : B(b) \Longrightarrow \mathfrak{K} \models_\epsilon j : H(h) \tag{5.2}$$

The result of applying the pattern (5.2) to DDLs describing two populated ontologies can be depicted graphically as given on Figure 5.6.

According to the semantics of individual correspondence, $i : b \mapsto j : h$ means that the pair of individuals $\left\langle b^{\mathcal{I}_i}, h^{\mathcal{I}_j} \right\rangle$ belongs to the domain relation r_{ij}. On one hand, to have a complete set of individual correspondences a knowledge engineer should take all individuals x of ontology i, find all semantically corresponding individuals y in ontology j, and explicitly specify that $i : x \mapsto j : y$ (extensional approach). On the other hand,

one can imagine a more compact definition, when an engineer specifies a transformation relation f_{ij} that allows to transform a set of individuals from i to individuals of j such that their interpretations respect the semantics of domain relation r_{ij}. In this later option, we can reformulate the pattern (5.2):

Given a distributed knowledge base $\mathfrak{K} = \langle \mathfrak{T}, \mathfrak{A} \rangle$ and a set of transformation relations f_{ij}, if $\mathfrak{B}_{ij}$ contains an into-bridge rule $i : B \xrightarrow{\sqsubseteq} j : H$, then:

$$\mathfrak{K} \models_\epsilon i : B(b) \Longrightarrow \mathfrak{K} \models_\epsilon j : H(f_{ij}(b)) \tag{5.3}$$

Consequently, we can give another reading to Figure 5.6. I.e., instance h was injected into A-box $\mathcal{A}_2$ with an assertion $2 : H(h)$ as a result of propagation via transformation of assertion $1 : B(b)$ from $\mathcal{A}_1$ along the into-bridge rule $i : B \xrightarrow{\sqsubseteq} j : H$.

It should be stressed again that a transformation relation f_{ij} plays a similar role on the level of individuals in A-boxes that the domain relation r_{ij} plays on the level of interpretation domains. However, since DL interpretation function maps individuals into elements of domain we can roughly think of the transformation relation as of a partial specification of the domain relation.

5.4.2 Distributed Instance Retrieval in DDLs

Given a distributed knowledge base $\mathfrak{K} = \langle \mathfrak{T}, \mathfrak{A} \rangle$, a *distributed retrieval of instances* of a concept D in ontology i is a task of finding all individuals x that instantiate D, i.e., $\mathfrak{K} \models_\epsilon i : D(x)$. For the sake of simplicity, we will consider only the possibility to retrieve instances of named concepts, leaving for the future work dealing with instance retrieval of complex concept expressions.

Our proposal consists in building an instance retrieving mechanism for DDLs on top of standard DLs instance retrieval algorithms. We will describe the intuition of the algorithm for the case of distributed knowledge base with only two constituent ontologies, i.e., when $\mathfrak{K} = \langle \mathfrak{T}, \mathfrak{A} \rangle = \langle \langle \{\mathcal{T}_1, \mathcal{T}_2\}, \mathfrak{B}_{12} \rangle, \langle \{\mathcal{A}_1, \mathcal{A}_2\}, \mathfrak{C}_{12} \rangle \rangle$. Due to the direction of bridge rules from 1 to 2, we will denote ontology $\langle \mathcal{T}_1, \mathcal{A}_1 \rangle$ as a *source* ontology, and $\langle \mathcal{T}_2, \mathcal{A}_2 \rangle$ as a *target* ontology.

We will also put some restrictions on a structure of A-boxes $\mathcal{A}_1, \mathcal{A}_2$ admitting only those logics that enjoy the property of reduction reasoning with instances to a pure terminological reasoning (see [97, 96, 50] for more details). This will allow us to complete the part of local instance retrieval with simple sound and complete instance retrieval algorithms defined for these logics.

The terminological propagation results in refining a concept taxonomy of target ontology according to the pattern (5.1). Hereafter, we will refer to a taxonomy built with respect to the terminological propagation as a *distributed taxonomy* and a reasoner capable of that functionality as a *distributed terminological reasoner*.

The assertional propagation reveals in either (a) asserting new concept membership for existing individuals of target ontology when a pattern (5.2) is used, or (b) injecting new individuals to target ontology in accordance with pattern (5.3) and further asserting their concept membership similarly to (a). In both cases, we will refer to such concept membership assertions as *distributed concept membership*.

The consequence of both propagational aspects affects distributed instance retrieval. The resulting instances can be naturally subdivided into two groups: ones that were explicitly defined in the target ontology A-box, and the others that were injected as transformation of instances of source ontology. Therefore, the distributed retrieval problem should be approached in a 3-step manner: (1) retrieve all local instances with respect to the distributed taxonomy of the target ontology, (2) retrieve all relevant instances of the source ontology and apply transformation to them, and finally (3) merge the results of two previous steps.

Algorithmically, the process of distributed retrieving the instances of named concept D in ontology $\langle \mathcal{T}_2, \mathcal{A}_2 \rangle$ w.r.t. a distributed knowledge base $\mathfrak{K}$ can be sketched as follows:

dRetrive$_2(D)$

1. compute a distributed taxonomy of $\mathcal{T}_2$
2. create a set $D_{\downarrow \mathcal{T}_2}$ of named concepts in $\mathcal{T}_2$ that are subsumed by a concept C w.r.t. a distributed T-box $\mathfrak{T}$ (these are equivalents and descendants of C in $\mathcal{T}_j$ computed w.r.t. $\mathfrak{T}$)
3. retrieve a set S^{local} of individuals in $\mathcal{A}_2$ using a standard DL instance retrieval algorithm with respect to the computed distributed taxonomy of $\mathcal{T}_2$
4. for every concept $H \in D_{\downarrow \mathcal{T}_2}$ connected via an into-bridge rule $i : B \xrightarrow{\sqsubseteq} i : H$
   (a) retrieve a set S_B of instances of B in ontology 1 invoking of **dRetrive**$_1(B)$
   (b) apply to elements of S_B a transformation function f_{12} and collect transformed instances in a set S_H
   (c) $S^{dist} \leftarrow S^{dist} \cup S_H$
5. return $S^{local} \cup S^{dist}$

The described algorithm can be generalized to a case of distributed knowledge bases with acyclical bridge graphs. Dealing with cycles opens additional questions, e.g., when to stop application of transformation for transformed individuals.

5.5 DRAGO Peer-To-Peer Reasoning Platform

In this section we will describe a design and implementation principles that lay in the base of DRAGO (Distributed Reasoning Architecture for a Galaxy of Ontologies), the platform for reasoning with multiple ontologies connected by semantic mappings.[2]

As depicted in Figure 5.1, DRAGO envisages a Web of ontologies being distributed amongst a peer-to-peer network of *DRAGO Reasoning Peers* (DRP). The role of a DRP is to provide reasoning services for ontologies registered to it, as well as to request reasoning services of other DRPs when this is required for fulfillment of distributed reasoning algorithm. The key issue of the DRP is that it provides possibility to register not just a stand alone ontology, but an ontology coupled with a set of semantic mappings.

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

Similarly to the process of attaching mappings to ontology, a DRP allows for attaching instance transformation relations, so that during the distributed instance retrieval the individuals from heterogeneous domains could be correctly transformed.

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

5.5.1 Architecture

A DRP constitutes the basic element of DRAGO. The major components of a DRP are depicted in Figure 5.7.

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

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

[2] `http://trinity.dit.unitn.it/drago`

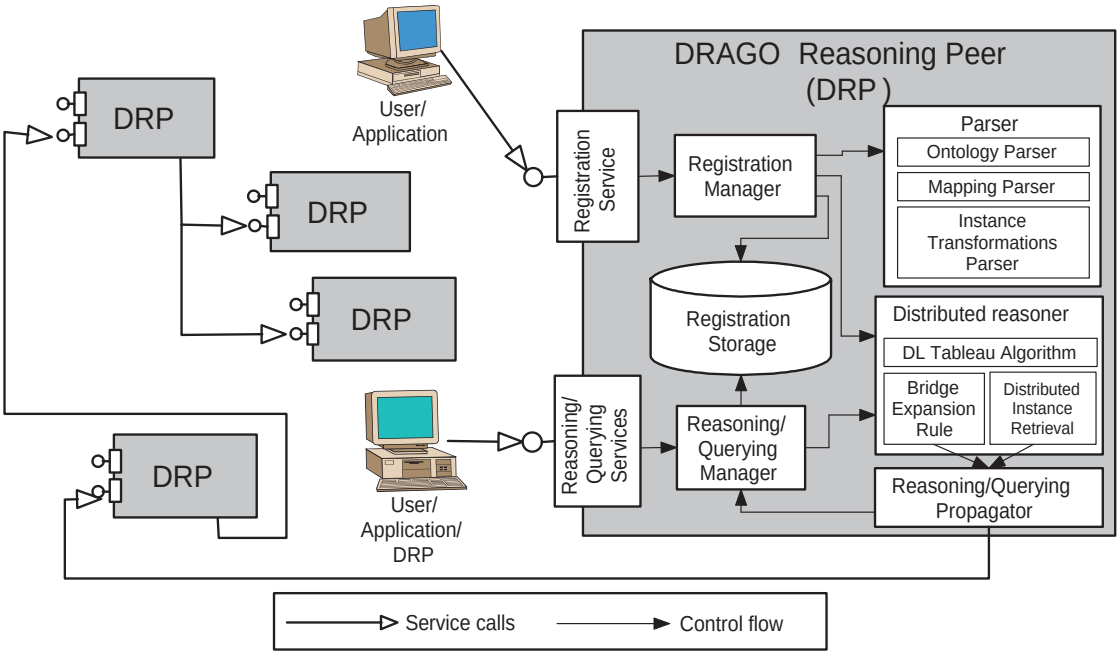


Figure 5.7: DRAGO architecture.

from defined (atomic) concepts.

All reasoning/querying services are divided into to groups: local and distributed. Local services are handled by a standard tableau reasoner, while distributed, by a distributed tableaux.

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

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

- URI to which the ontology will be bound
- URLs of ontology and semantic mappings attached to it, if any
- if the semantic mappings connect this ontology with ontologies registered to external DRPs then additionally the URLs of these DRPs should be specified. This requirement is explained by the necessity to know who is responsible for reasoning with these ontologies.
- when dealing with heterogeneous individuals, a DRP should be also provided with instance transformation specification.

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

The *Parser* component translates ontologies, mappings and instance transformations source files to the internal format used by the Distributed Reasoner. For doing so, the Parser consist from two sub components: the ontology parser, tailored on ontology language formats (for example, OWL [9]), the mapping parser, tailored on mapping formats (for example, C-OWL [11]), and parser of instance transformations (in XML format, see Figure 5.9 for a sample code).

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

The *Distributed Reasoner* represents a brain of a DRP. It realizes the distributed reasoning and querying algorithms sketched in this chapter and reasons on ontologies with attached mappings and instance transformations that are registered to the DRP. The Distributed Reasoner is built on top of standard tableau reasoner whose algorithm was extended with the additional Bridge Expansion Rule in accordance with the distributed tableau algorithm. Distributed tableaux was also extended to handle instance retrieval requests in accordance with the distributed instance retrieval algorithm. When the Bridge Expansion Rule is applied it analyzes semantic mappings and possibly generates reasoning subtasks that are required to be executed in the ontologies participating in the mappings. Similarly, the querying subtasks can be possibly generated.

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

5.5.2 Implementation

The described DRAGO architecture has been implemented for the case of OWL [9] ontology space. For expressing semantic mappings between OWL ontologies we use a C-OWL [11, 12]. According to C-OWL, mapping consists of references to the source and target ontologies and a series of bridge rules relating classes between these ontologies (see Figure 5.8 for a sample extract). Among the possible C-OWL bridge rule types DRAGO supports the use of $\equiv, \sqsubseteq, \sqsupseteq$ rules connecting defined(atomic) concepts.

A Distributed Reasoner was implemented as an extension to an open source OWL reasoner Pellet [34]. Originally, Pellet parses OWL ontology to a Knowledge Base (T-box/A-box). To satisfy the needs of DRAGO we extended a Pellet's Knowledge Base with an M-box containing parsed C-OWL mappings. Other extensions of Pellet were done by adding a Bridge Expansion Rule to the core tableau algorithm in order to transform it to the distributed tableau and extending core instance retrieval mechanism of Pellet reasoner. The Bridge Expansion Rule rule is called for every node created by the core tableau algorithm and consist in finding such bridge rules in the M-box that are capable of importing new subsumptions from mapping-related ontologies.

DRAGO is implemented in java and consist of two core packages: (1) a reasoning peer, DRP, and (2) a client, called DRPConnector. Both applications have opened API so that external java-applications can easily invoke (start/stop/reason/query) DRAGO functionality.

DRP plays a role of reasoning server. Being started, it opens a socket on a specified port and listens for incoming reasoning/querying requests. Requests are send/received over the TCP/IP network in accordance with a simple text-based protocol. DRP is a multi-threaded application, thus it can service multiple simultaneous requests from users. To simplify the communication with DRP and "hide" from a user unnecessary awareness of the communication protocol we have developed the DRPConnector application, which serves as a middleware between user and a certain reasoning server. Moreover, a DRP itself utilizes DRPConnector functionality in a Reasoning/Querying propagator block.

Starting multiple DRPs on different hosts forms a DRAGO network of distributed reasoning peers.

5.5.3 Working example

Let us step-by-step "run" a working example to get the impression of using DRAGO. Consider again the illustrative scenario that we have given in the beginning of Section 5.4. Two human resource offices, of a university and of one of its department, utilize ontologies for describing research staff, the university uses O^{Uni}, while the department O^{Dept}. The idea is that the university wants to re-use knowledge that exists in the department.

To run this idea, we associate to the university human resource office a reasoning peer DRP^{Uni}, and to a department office a DRP^{Dept}. Further, the following steps need to be performed:

- assign ontology O^{Dept} to DRP^{Dept}
- define semantic mapping $M^{Dept-Uni}$ between overlapping concepts of ontologies O^{Dept} and O^{Uni} utilizing C-OWL format (see a sample extract on Figure 5.8). For example, we can have the statement that

$$
\begin{aligned}
O^{Dept} : PersonContact &\xrightarrow{\sqsubseteq} O^{Uni} : PersonContact \\
O^{Dept} : Article &\xrightarrow{\sqsupseteq} O^{Uni} : ConferencePaper \\
O^{Dept} : PhDThesis &\xrightarrow{\equiv} O^{Uni} : DoctoralThesis
\end{aligned}
$$

- define individual level transformations $f^{Dept-Uni}$ (see a sample extract on Figure 5.9). For example, we can have the statement that

```
<domain-relation sourceOntology="O^Dept">
   <instance-transformation
      sourceConcept="O^Dept : PersonContact"
      value="${function.JavaEmailExtractor}"
```

  where `JavaEmailExtractor` is a function in java class that is capable of downloading the web pages, parsing the text and extracting from it e-mail addresses.

- assign ontology O^{Uni} with mapping $M^{Dept-Uni}$ and instance transformation $f^{Dept-Uni}$ to DRP^{Uni}
- notify the DRP^{Uni} that ontology O^{Dept} in $M^{Dept-Uni}$ and $f^{Dept-Uni}$ is supported by DRP^{Dept}
- start the peers O^{Uni} and O^{Dept}.

From that moment, both of the reasoning peers are ready to respond to reasoning and querying questions (see a sample extract of DRAGO API invocation in java on Figure 5.10).

As we have described in Section 5.5, every DRP is capable of providing local and distributed reasoning and querying services. Local services "ignore" semantic mappings (if any) and consider only local axioms of ontology. In turn, the distributed services depend on axioms that propagate via mappings and instances that propagate via combination of mappings with the specified instance transformations. To distinguish these services we will use letters "*L*" to denote local, and "*D*" to denote distributed reasoning and querying services. In particular, you can see the sample invocation of several services on Figure 5.10, such as *getLSubClasses*, *getDSubClasses*, *getLTaxonomy*, *getDTaxonomy*, *getLInstances*, *getDInstances* and etc.

5.6 Applications

The main application domain of proposed in DRAGO peer-to-peer reasoning and querying paradigm comes out from its seamless integration with semantic matching techniques

[13, 76, 35, 33]. Indeed, the goal of semantic matchers is to produce semantic relations between heterogeneous ontologies, whereas the goal of DRAGO is to exploit these mappings for enabling distributed reasoning and querying. Having combined semantic matching techniques with distributed reasoning and querying techniques will allow to create a full-fledged platform for resolving ontological heterogeneity problems on the semantic web.

In this section, we describe several intuitive applications of DRAGO. In particular, we sketch the problems of modular ontology reasoning, verification of semantic mappings, deriving new mappings, and finally, assisting ontology development.

5.6.1 Modular Ontology Reasoning and Querying

The semantic web can be rationally assumed to contain multiple distributed ontologies, modules, and the modularization, therefore, can be seen as a mechanism for assembling some of these modules into a coherent network that can be referred to as a single entity, a modular ontology. Utilizing semantic mappings as glue for connecting autonomous ontological modules and then applying DRAGO to such a setting, we get the possibility of composing a modular ontology.

5.6.2 Reasoning about Mappings

As we have already seen, semantic mappings are very important for enabling reasoning and querying over heterogeneous ontologies. However, the mappings used should be of good quality. This can be achieved in certain cases when mappings are established manually by a knowledge engineer. However, automatic matchers suffer from mapping imperfectness. To improve the quality of mappings one needs a mechanism of reasoning over mappings themselves. In [92] authors make a first step towards shedding the light on this problem grounding their approach on top of Distributed Description Logics and implementing it on top of DRAGO.

5.6.3 Semantic Mappings Verification

The problem of mapping verification can be roughly considered as giving response on the question how good is certain mapping produced by any semantic matcher. Following a simple idea that of applying to that mapping a distributed reasoning technique and further checking which concepts due to the mappings become unsatisfiable allows to built a mechanism of detecting defective mapping.

5.6.4 Ontology Development Assistance

Ontology development process is difficult and time consuming task. Due to that the OWL language proposed a mechanism of "importing" already existent ontologies allowing to reuse previously defined knowledge. However, the import mechanism allows for only importing the whole ontology, whereas in many situations one would like to selectively import specific knowledge. For that reason, we envisage the following ontology development scenario. A knowledge engineer defines (enumerates) key concepts of ontology he needs to create. Afterwards, running some (semi-)automatic matcher he establishes semantic links to different preferred ontologies which describe the similar problem domain. Applying then to that setting DRAGO reasoning technique allows reifying initial enumeration of key concepts into, for example, a hierarchically ordered taxonomy.

5.7 Conclusions and Future Directions

In this chapter, we described the main theoretical and practical issues of distributed reasoning on the semantic web. First, we did an overview of Distributed Description Logics framework (DDLs) that is capable of representing multiple heterogeneous ontologies. Afterwards we clarified the propagational driving forces (patterns) of reasoning and querying in DDLs. Having the propagational patterns allowed us then to define distributed algorithms for basic reasoning and query answering services, such as checking concepts satisfiability and retrieving the set of individuals that instantiate a certain concept. We finalized the discourse with describing the architectural and implementation aspects of DRAGO distributed reasoning platform and highlighted several possible applications of DRAGO to different problems on the semantic web.

As a future work, we plan to extend the DDLs framework with a reason-able ability to support complex mappings involving nominals, and to enable distributed answering not just instance retrieval queries over defined concepts but over arbitrary concept expressions. From the practical side, we will pursue the implementation in DRAGO of the ideas identified in the Application section, in particular integration with semantic matchers, performing mappings verification and reasoning about mappings.

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:cowl="http://www.itc.it/cowl#"
    xmlns="http://example#" xml:base ="http://example">
 <cowl:Mapping>
   <cowl:sourceOntology>
     <owl:Ontology rdf:about="http://dept.owl"/>
   </cowl:sourceOntology>
   <cowl:targetOntology>
     <owl:Ontology rdf:about="http://uni.owl"/>
   </cowl:targetOntology>
   <cowl:bridgeRule>
     <cowl:Into>
       <cowl:source>
         <owl:Class rdf:about="http://dept#PersonContact"/>
       </cowl:source>
       <cowl:target>
         <owl:Class rdf:about="http://uni#PersonContact"/>
       </cowl:target>
     </cowl:Into>
   </cowl:bridgeRule>
   <cowl:bridgeRule>
     <cowl:Onto>
       <cowl:source>
         <owl:Class rdf:about="http://dept#Article"/>
       </cowl:source>
       <cowl:target>
         <owl:Class rdf:about="http://uni#ConferencePaper"/>
       </cowl:target>
     </cowl:Onto>
   </cowl:bridgeRule>
   <cowl:bridgeRule>
     <cowl:Equivalent>
       <cowl:source>
         <owl:Class rdf:about="http://dept#PhDThesis"/>
       </cowl:source>
       <cowl:target>
         <owl:Class rdf:about="http://uni#DoctoralThesis"/>
       </cowl:target>
     </cowl:Equivalent>
   </cowl:bridgeRule>
   ...
 </cowl:Mapping>
</rdf:RDF>
```

Figure 5.8: Extract of sample mapping in C-OWL format

```
<?xml version="1.0"?>
<drago-dr-config>
  <domain-relations>
    <domain-relation sourceOntology="http://dept.owl">
      <local-functions>
        <function
          name="JavaEmailExtractor"
          cls="it.unitn.drago.JavaEmailExtractor.class"
         >
          <params>
            <param type="String"/>
          </params>
        </function>
      <instance-transformations>
        <instance-transformation
          sourceConcept="http://dept#PersonContact"
          value="function.reverseString{param.sourceInstance}"
          description="Extract e-mail visit www homepage"/>

        ...

      </instance-transformations>
    </domain-relation>
  </domain-relations>
</drago-dr-config>
```

Figure 5.9: Extract of instance transformations specification in XML format

```
// starting DRP for source ontology
DRP drpSource = new DRP(4444);
drpSource.setOntology("http://dept.owl", "onto/dept.owl");
drpSource.load(); drpSource.startDRP();

// starting DRP for target ontology + mapping
DRP drpTarget = new DRP(5555);
drpTarget.setOntology("http://uni.owl", "uni.owl");
drpTarget.setMapping("http://dept-uni.cowl", "onto/dept-uni.cowl");
drpTarget.setSupportedBy("http://uni.owl", "localhost:4444");
drp5555.setInstanceTransformations("onto/dept-uni.dr");
drpTarget.load(); drpTarget.startDRP();

// starting Client to access reasoning services for target ontology
DRPConnector client = new DRPConnector();
client.connect("localhost:5555");
// check local consistency of target ontology
client.isLConsistent());
// check distributed consistency of target ontology
client.isDConsistent());

// get all local subclasses of Article concept
client.getLSubClasses("http://uni#Article"));
// get all distributed subclasses of Article concept
client.getDSubClasses("http://uni#Article"));

// get all local superclasses of Employee concept
client.getLSuperClasses("http://uni#Employee"));
// get all distributed superclasses of Employee concept
client.getDSuperClasses("http://uni#Employee"));

// compute local concept hierarchy (use only local axioms)
client.getLTaxonomy());
// compute distributed concept hierarchy (+ propagated axioms)
client.getDTaxonomy());

// retrive local instances (use only local axioms)
client.getLInstances("http://uni#PersonContact")
// retrive distributed instances (+ propagated axioms + transformed instances)
client.getDInstances("http://uni#PersonContact")
      ...
client.disconnect();

// Stop reasoning peers
drpTarget.stopDRP();
drpSource.stopDRP();
```

Figure 5.10: Extract of sample java code of DRAGO API usage

Chapter 6

Efficient Distributed Information Retrieval based on Classification and Content

by WOLF SIBERSKI

6.1 Introduction

In this chapter, we treat the integration of Ontology-based querying and classic information retrieval methods[1]. In practical Semantic Web applications, resources such as Web pages often have associated document metadata (resource descriptions), according to some ontology, *and* some document content which is not available through reasoning approaches. In case of text-based content, information retrieval provides proven and reliable algorithms for identification of relevant documents, given user-specified search terms. Of course, in the Semantic Web we need an approach which provides search capabilities in a network of information sources.

Peer to peer systems are a powerful paradigm to address some of these problems. Not relying on central coordination federations of information sources are formed dynamically by independent nodes. At any time new sources can join the network and disseminate their documents in a more timely way than the crawling of central servers can ascertain. File sharing applications, where media files are retrieved based on simple meta-data annotations like file formats or names, have become increasingly popular due to their ease of use. Also digital library collections can benefit from the advantages of P2P infrastructures, since much of their content (like embedded images) can be annotated by meta-data.

[1]Part of this work has been presented at the International Semantic Web Conference 2004 [74], the International Conference on Data Engineering 2005 [7], and the European Conference on Research and Advanced Technology for Digital Libraries 2005 [6]

However, flexible search based on classification and content is a challenge in a P2P environment. On one hand this is because queries have to be evaluated over the network at search time, which in basic file sharing applications is usually done by flooding queries through the complete network (or at least within a certain radius). On the other hand almost all effective textual measures for information retrieval not only rely on statistics about the single documents, but also integrate statistics on the entire collection of all documents, e.g. how well individual keywords discriminate between documents with respect to the entire collection (inverted document frequencies). This so-called collection wide information cannot be derived locally.

To improve query efficiency techniques one way are central indexes and distributed hash tables (DHTs)[91, 81, 1]. Besides the speed up above naive query flooding an additional advantage is that by such a structure also collection wide information can be provided for subsequent querying. A major drawback is that such indexes use up a lot of the available bandwidth by the necessary administrative message exchange for upkeep, because every change in the federated document collection (e.g. content changes within some peer) has to be registered in the index. A contrasting way of gaining query efficiency are local routing indexes that avoid the overhead of constant index upkeep, but due to their local nature face problems with acquiring the necessary collection-wide information. Besides having to be efficient, querying schemes will also have to take into account that the information in digital libraries often is pre-structured. Libraries usually categorize documents following some standardized taxonomies, such that documents on similar topics might be distinguished e.g. by rather taking an economical or sociological point of view, etc. This structure is also used to sometimes resolve ambiguities of keywords.

In this chapter we investigate the querying of federated information sources over a peer-to-peer network. But in contrast to central indexing schemes, our aim is to create a local indexing scheme that allows effective indexing with a minimum of management message overhead and even efficiently use collection-wide information. Moreover, we will exploit taxonomies to structure the individual collections and investigate how this pre-structuring interacts with the effectiveness of our local indexing scheme and how to deal with the trade-off between the total number of documents in each category and our respective index size. We have measured the necessary message traffic, the quality of result sets (as opposed to the perfect results using a central index), and a number of other characteristics of our novel approach. For our work, we assume a strong cooperation between peers, e.g. in order to ensure consistency of ranking results, score values have to be calculated uniformly by all peers.

We motivate our approach with the example of federated news collections. News items can also be compound documents and are usually categorized within certain topics like politics or sports. Since we want to focus on the textual retrieval, we use a collection of LA Times news articles from the TREC-5 collection for our evaluations. By assuming periodic changes of user interests we can also experiment with the arrival or removal of complete corpora from our federation. Since queries in such practical applications usually form a Zipf distribution (i.e. considering the total set of queries very little queries are

posed very often, whereas most queries are posed only once in a while), we will present extensive experiments for such a distribution. Given our innovative results, peer-to-peer networks are on the verge of forming efficient infrastructures for federations of digital libraries utilizing even collection-wide information without the communication overhead of central indexes.

6.2 Information Retrieval in a Distributed Environment

In large document collections information retrieval techniques are mandatory for efficient retrieval. Over centralized repositories these techniques have been investigated since the 70ies and work quite effective, e.g. using inverted file indexes for subsequent retrieval [63]. Maintaining these indexes, however, is a major problem in distributed systems, especially peer-to-peer networks that often share vast numbers of documents and have a high volatility with respect to peers joining and leaving the network. In contrast to static document collections every peer joining or leaving the network registers its document collection or removes it, thus indexes have to be updated very often.

For local query evaluation schemes a particular problem arises when collection wide information is an integral part of the query processing technique. For instance in the case of TFxIDF [107] the term frequency may be locally evaluated for each specific document, however, for the document frequency a snapshot of the entire current content of all active peers needs to be evaluated. Of course this would immediately annihilate any benefits gained by sophisticated local querying schemes.

Consider a simple example to show how local scorings fail, if collection-wide information has to be considered in the retrieval process. Assume the case that we have just two peers that should return their best matches with respect to the most popular information retrieval measure TFxIDF. This measure is a combination or two parts, the term frequency (TF, measures how often a query term is contained in a certain document), and the inverted document frequency (IDF, inverse of how often a query term occurs in the document collection). This measure needs to integrate collection-wide information and cannot be determined locally.

As an instance take a simple conjunctive query Q for the terms 'a' and 'b' posed to two peers P_1 and P_2 that has to be evaluated locally at each peer. Let's assume that P_1 contains three documents D_1, D_2 and D_3, and P_2 also contains three documents D_4, D_5 and D_6. For simplicity of our example let us further assume that in D_1toD_6, our two keyword occur mutually exclusive in the documents: D_1, D_2 and D_6 contain the keyword 'a', whereas D_3, D_4 and D_5 contain the keyword 'b'. Moreover, assume that all documents are of the same length and the keywords occur in the same number in all documents, such that the respective term frequency is the same for all documents. Evaluating our query Q locally we have now to rank the documents in each peer. Since the keywords are mutually exclusive in our document base and the TFs are equal for each document, the ranking is

only determined by the weighting factor of the occurring term in the IDF.

In peer P_1 we have two documents out of three containing term 'a', i.e. an IDF of $\frac{3}{2} = 1.5$, and only one document containing 'b', i.e. an IDF of $\frac{3}{1} = 3$. That means that with respect to Q P_1 ranks D_3 as better than D_1 and D_2. Symmetrically P_2 ranks D_6 higher than D_4 and D_5, because here 'b' occurs in two documents and 'a' only in one. Integrating the results from $P1$ and $P2$ we get a higher ranking of D_3 and D_6 than of the four other documents. In contrast, performing query Q over a central collection containing all six documents D_1 to D_6, we find that both query terms 'a' and 'b' occur in three of the six documents, i.e. have an IDF $\frac{6}{3} = 2$. Since the TF is still the same, all six documents will be correctly assigned the same score.

As shown, collection-wide information is essential to provide proper document scores. But the index holding this information does not necessarily need to be completely up-to-date; obviously there is a trade-off between index information that is 'still current enough' given the network volatility and the accuracy of the query results. Research on what dissemination level is required in Web IR applications to allow for efficient retrieval showed that a complete dissemination with immediate updates is usually unnecessary, even if new documents are included into the collection [102]. Moreover, the required level was found to be dependent on the document allocation throughout the network [101]: random allocation calls for low dissemination, whereas higher dissemination is needed if documents are allocated based on content. Thus a lazy dissemination usually has comparable effectiveness as a centralized approach for general queries, but if only parts of the networks containing most promising documents with similar content are queried, the collection-wide information has to be disseminated and regularly updated.

6.3 Approach

As shown in the previous section, we need collection-wide information at each peer to do a correct score computation. The challenges are

- how to compute this information
- where to store it, and
- how to distribute it in the network.

The key to success is the observation that we don't need a complete inverted index to process a query. For example, to calculate the correct scores, peers P_1 and P_2 need only IDFs for terms a and b, but not for all terms occurring in their documents.

Storage To store this information, we use a super-peer network approach: in a P2P-network peers often vary widely in bandwidth and computing power. As discussed in

[108] exploiting these different capabilities can lead to an efficient network architecture, where a small subset of peers, called super-peers, takes over specific responsibilities. In our case, we assign the index management responsibility to the super-peers. The super-peers form the network backbone, and each document provider peer is directly connected to one of them.

Distribution A query consists of a category of the taxonomy which should be searched and a conjunction of keywords that are searched in the content of the documents. Before distributing such a query, a super-peer adds the necessary collection-wide information from its index to it. If it isn't yet in the index, an estimation is provided.

The category in the query is used as a filter for two purposes: First to reduce the number of peers (and thus documents) which must be searched and ranked. Second the user can use the category to avoid ambiguities. If a user is interested in the local sport-team called 'Jaguars', the appropriate category will avoid hits Jaguar-cars and the animal Jaguar. The keywords which are specified in the query will only be used for the documents which are in the named category.

Computation Responding peers do not only deliver matching documents, but also each add local data necessary to compute the collection-wide information (for TFxIDF this is the document frequency for each query term and the document count). On the way back to the originating super-peer, this data is aggregated. Thus, the originating super-peer gets everything it needs to compute the complete aggregate, and can store the computed result in its index.

The next subsections discuss in detail how this approach is applied to the digital library network context.

6.3.1 Query Processing

Query distribution at super-peer Each super-peer maintains an IDF index containing IDF values for the keywords. This is done separately for each category, not for all documents in all categories. Thus, the key for this index is built from a category and one keyword. As mentioned above a query contains the IDF values for the query-terms. The IDFs are taken from the IDF index, or estimated if a keyword is not yet in the index. In the latter case, the average IDF is used as estimation.

Query processing at peer At each peer, first only documents in the specified category are taken into account. The top-k documents are determined using the TFxIDF algorithm, but based on the IDF values from the query. If this gets enough (=k) results in the queried category, these are sent to the super-peer. If the number of documents matching the query is smaller than k, the query is relaxed, first to subcategories and then to

the super-categories. This process is repeated until the peer has k results or the root of the taxonomy is reached. The entries found in all newly searched categories are sorted by their similarity to the originating category using the following measure (from Li et al. [65]):

$$sim(c_1, c_2) = e^{-\alpha l} \cdot \frac{e^{\beta h} - e^{-\beta h}}{e^{\beta h} + e^{-\beta h}}$$

where l is the shortest path between the topics in the taxonomy tree and h is the depth level of the direct common subsumer. α and β are parameters to optimize the similarity measurement (best setting is usually $\alpha = 0.2$ and $\beta = 0.6$).

The super-peer then gets the top-k of the peer or a number $n < k$ of documents matching. This query-relaxation is shown in the following code:

```
Initialize a ResultSet results;

Set searchRoot := Category from query

do
  Initialize a new set searchCategories
  Add searchRoot to searchCategories
  while (number of results < k and searchCategories is not empty
  begin
    Initialize new set allChildren
    for all cat in searchCategories do
    begin
      // retrieve hits matching category exact
      Initialize ordered list matchingDocuments
      for all doc in documents
      begin
        if (document-category = cat
           AND document contains terms from query
           AND number of matching documents < k
           OR doc.score(query.terms) > matchingDocuments.getLastDoc.score))
           then add document to matchingDocuments
      end
      results.addHits(retrieveExact(cat, query))
      allChildren.add(cat.children);
    end

    searchCategories = allChildren;  // go one level down in category tree
    removesearchRoot from searchCategories; //do not to traverse subtree twice
  end

  searchRoot := parent of searchRoot // go one level up in category tree
while(not k results AND searchRoot != nil);

trim results to k  // in case we collected more than k entries

return results;
```

Result merging at super-peer A super-peer retrieves max. k hits from each of its peers and combines them to the top-k. As described above it is possible that peers also send results which are coming from another category as requested. In this case, the super-peer first takes all hits which match the queried category. If this results in a set smaller than k it

takes the next best-matching hits from each peer and combines them using the described sorting.

IDF index update The IDF index can be updated in two ways:

1. By summing up document frequencies and document counts delivered from connected peers and super-peers, the super-peer where the query originated computes IDFs for each query term and updates its IDF index. If the difference between computed IDF and estimated IDF value exceeds a threshold, the query is redistributed, this time using the computed IDF values.

2. if a super-peer receives a query it checks, if the IDFs contained are marked as estimated. If this is not the case, these values are used to update the IDF index.

6.3.2 IDF index entry expiration

Viles and French have shown that in a large document collection IDF values change slowly [102]. In our context, this is not strictly applicable, because there are two kinds of changes that may influence our collection-wide information significantly:

1. *New documents with similar content: new peers join the network.*
   Imagine a large federation of library servers which offer articles from different newspapers. Let's assume we already have a newspaper like the NY Times in the collection. What can happen if peers join the network offering a new newspaper, i.e. the LA Times? In this case we can be sure that the articles usually will be on nearly similar topics except a few local news. Thus, we do not really have to update our IDFs since the words in the articles are distributed the same way as before.

2. *New documents or new corpora: New library servers join the federation or new documents are included in existing collections, whose content is very different from existing articles and thus shifts IDFs and changes the discriminators.*
   Let's look at an example: Assume there is an election e.g. in France and people use our P2P-news-network to search for news regarding this election. This normally will be done using queries like 'election France' and results in a list of news that contain these words. In this case there would be a lot of news containing France, thus 'election' is the discriminator, and the IDFs will give us the correct results. Now think of another election taking place in the US in parallel. The term 'election' will no longer be the best discriminator, but the term 'France' then gets more important.

In these cases we have to solve the problem that entries in the IDF index become outdated over time. We can handle both cases in the same way: Each IDF value gets a

timestamp when the term appears for the first time and the term/IDF-pair is stored. After a specific expiration period (depending on the network-volatility) the item becomes invalid and the entry gets deleted. In this way we force IDF recomputation if the term occurs again in a query. By adjusting the expiration period we can trade off accuracy against performance. We reduce the expiration period for terms occurring more frequently, thus ensuring higher accuracy for more popular queries.

6.3.3 Query Routing Indexes

So far, we still distribute all queries to all peers. We can avoid broadcasting by introducing additional routing indices which are used as destination filters:

- For each category in our taxonomy the *category index* contains a set of all peers which have documents for this category. It is not relevant if this peers did contribute to queries in the past.

- In the *query index* for each posed query the set of those peers which contributed to the top-k for the query are stored.

Query Distribution The super-peer first checks if all query terms are in the IDF index. If this is not the case the query has to be broadcast to permit IDF aggregation. We also broadcast the query if none of the routing indexes contain applicable entries.

If an entry for query exists in the query-index, it is sent to the peers in this entry only, since no other have contributed to the top-k result for the current query.

Otherwise, if the query category is in the category index, the query is sent to all peers to the corresponding category entry.

Index Update For each delivered result set, a query index entry is created, containing all peers and super-peers which contributed to the result.

For the category index, we need to know all peers holding documents of the specified category, even if they didn't contribute to the result set. Therefore, we collect this information as part of the result set, too, and use it to create category index entries.

As with the IDF index, the network volatility causes our routing indexes to become incorrect over time. We use the index entry expiration approach here as well to delete outdated entries.

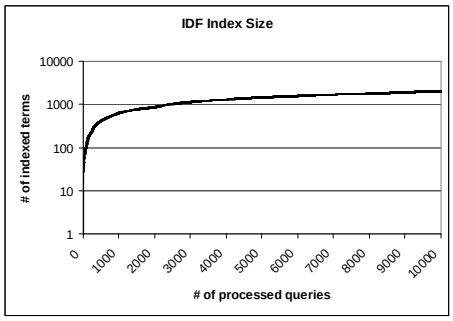


Fig. 1. Index size

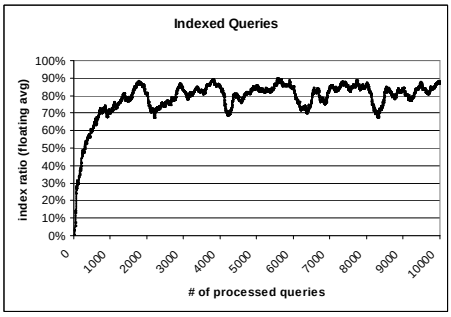


Fig. 2. Coverage of query index

6.4 Evaluation

6.4.1 Simulation Environment

We use the TREC document collection volume 5 consisting of LA Times articles for our experiments. The articles are already categorized according to the section they appeared in, and we use this information as base for our document classification. To simulate a network of document providers, these articles are distributed among the peers in the network. The simulated network consists of 2000 peers, each providing articles from three categories on average (with a standard deviation of 2.0).

The simulation is based on the framework described in [89]. The super-peers are arranged in a HyperCuP topology [85]. The TFxIDF calculation based on inverse indexes was done using the (slightly modified) search engine Jakarta Lucene [2].

We assume a Zipf-distribution for query frequencies with skew of -0.0. News articles are popular only for a short time period, and the request frequency changes correspondingly. With respect to the Zipf-distribution this means that the query rank decreases over time. Query terms were selected randomly from the underlying documents. In our simulation, we generate 200 new most popular queries every 2000 queries which supersede the current ones and adjust query frequencies accordingly. This shift may be unrealistically high, but serves well to analyze how our algorithm reacts to such popularity changes.

6.4.2 Results

Index size Figure 1 shows how the IDF index at each super-peer grows over time. After 10000 queries it has grown to a size of 2015, only a small fraction of all terms occuring in the document collection. A global inverted index we would have had contained 148867 terms. This underlines that much effort can be saved when only indexing terms which are actually appearing in queries.

[2] http://jakarta.apache.org/lucene/docs/index.html

Index effectivity Both category and query index become quite effective. After nearly 2000 queries, the query index achieves a coverage of 80%. Figure 2 shows how each popularity shift causes a coverage reduction from which the query index recovers after about 1000 queries. This shows that a change in query popularity over time is coped with after a very short while.

As there are only about 120 different categories, after less then 1000 queries the index contains nearly all of them (Figure 3). We assume that news provider specialized on some topics change these topics only very infrequently. Therefore, peers do not shift their topics during the simulation. Thus, the category index serves to reduce the number of contacted peers continuouosly, also after popularity shifts.

Figure 4 shows how many peers had to be contacted to compute the result. The influence of popularity shifts on the whole outcome can also be seen clearly. The category index takes care that the peaks caused by popularity shifts don't become too high. Summarized, the combination of both indexes yields a high decrease of contacted peers compared to broadcasting.

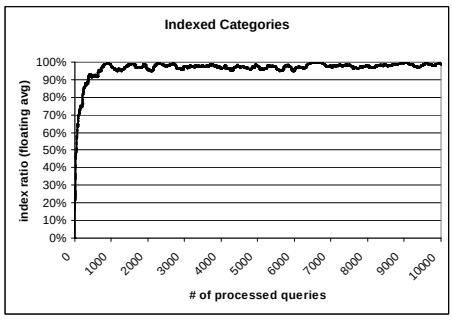


Fig. 3. Coverage of category index

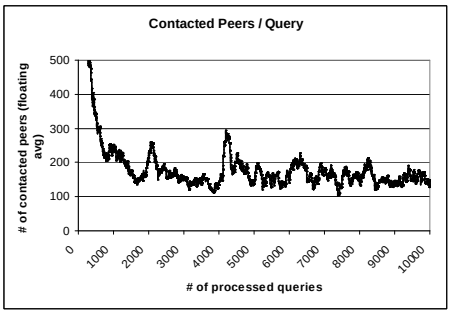


Fig. 4. Contacted peers per query

In the experiments described here we didn't introduce dynamics regarding the peers contents. Therefore, our algorithm yields exactly the same results as a complete index. In [7] (where we didn't take categories into account), we show that if 20% of the peers contents during a simulation run, the error ratio is about 3.5%.

6.5 Related Work

Since the concepts of the highly distributed P2P networks and the rather centralized IR engines are hard to integrate, previous work in the area is focusing on efficient dissemination of this information. There is a challenging trade-off between reduced network traffic by lazy dissemination however leading to less effective retrieval, and a large network traffic overhead by eager dissemination facilitating very effective retrieval. What is needed is "just the right" level of dissemination to maintain a "suitable" retrieval effectiveness. Thus previous approaches to disseminate collection-wide information rely

on different techniques. We will briefly review the techniques from peer-to-peer systems, from distributed IR and Web Search engines and compare them to our approach.

For peer-to-peer systems there are different approaches. The PlanetP system [19] does not use collection-wide information like e.g. the inverted document frequency of query terms directly, but circumnavigates the problem by using a so-called inverted peer frequency estimating for all query terms, which peers are interesting contributors to a certain query. Summarizations of the content in the form of Bloom filters are used to decide what content a peer can offer, which are eagerly disseminated throughout the network by gossiping algorithms. Thus in terms of retrieval effectiveness this scheme describes documents on the summarization level, which is a suboptimal discriminator and by gossiping the system's scalability is limited. The idea of PeerSearch [95] is comparable to our approach, but instead of a broadcast-topology CAN [81] is used in combination with the vector space model (VSM) and latent semantic indexing (LSI) to create an index which is stored in CAN using the vector representations as coordinates. Thus all collection-wide information has to be disseminated again leading to a limited scalability. Also summarizing indexes have been used to maintain global information about a set of documents like e.g. in [106]. Here so-called cell abstract indexes are used for approximate queries. The abstract of a set of documents is some statistics of all documents in the set and the abstract of a peer is an abstract of the shared document set residing in the peer. An abstract index of a P2P system then is an organization of all abstracts of peers in the system. All peers of a system can thus be formed into an overlay network. Every joining peer will be added to a cell that contains its abstract and subsequently queries are routed to those cells that contain their abstract. However, also in this case indexes for all cells have to be updated regularly leading to a high overhead of network traffic. Moreover, peers in the end cells will just deliver all documents to the querying peer not removing suboptimal objects and again causing unnecessary network traffic. As in our approach, [67] use super-peers (called "'hub"' nodes) to manage indices and merge results. Depending on the cooperation capability/willingness of document providers ("'leaf"' nodes), hub nodes collect either complete or sampled term frequencies for each leaf peer. This information is used to select relevant peers during query distribution. By using query sampling hubs are able to give an estimate of relevant peers, even in case of uncooperative peers. As with the other systems, indices are built in advance, thus causing possibly unnecessary management messages.

From the perspective of information retrieval the problem of disseminating collection-wide information first occurred when IR moved beyond centralized indexing schemes over collections like e.g. given by TREC, and had to deal with vast distributed document collections like the WWW. Here due to the random-like distribution of content over the WWW, research on effective retrieval in Web IR applications showed that a complete dissemination with immediate updates is usually unnecessary, thus allowing for a little volatility [102], The required level of dissemination, however, was found to be dependent on the document allocation throughout the network [101]: random allocation calls for low dissemination, whereas higher dissemination is needed if documents are allocated

based on content. In peer-to-peer networks this random-like distribution does usually not hold. We have argued in our news scenario, that in practical applications peers often will not carry a random portion of the entire document collection. Though some newspapers like the New York Times will cover a wide area of topics, specialized newspapers like the Financial Times will limit the range and some publications can even provide corpora that essentially differ in the topics and keywords contained. Moreover, though a lazy dissemination in terms of effectiveness usually is comparable to the centralized approach for general queries, our indexing scheme focuses only on parts of the networks containing most promising documents, thus the collection-wide information has to be disseminated and (at least) regularly updated. Hence, classical Web search engines like Google crawl the Web and individually index the sites, but then all indexing information is transferred over the network and managed in a vast centralized repository for subsequent retrieval. Novel approaches to distribute Web search engines like Google desktop will have to deal with the same problem of dissemination this information efficiently. Therefore, though designed for peer-to-peer infrastructures, our work here can be assumed to have an interesting impact on future developments in distributed Web search engines.

6.6 Summary

In this chapter we have discussed the important problem of efficiently querying federated information sources using peer-to-peer infrastructures especially if collection-wide information is needed. We have described a practical use-case scenario for the problem and have presented an innovative local indexing scheme which flexibly includes collection-wide information. Our novel indexes are not created in advance, but are maintained query-driven, i.e. we do not index any information which is never asked for. This allows our algorithm to scale, even in more volatile networks. Another improvement is our introduction of a separate category index that allows to prune large portions of the network and thus also enhances scalability.

In the next step, we plan to extend the simple classification scheme to a description logic based annotation scheme. This will allow more complex constraints on the metadata part of a query, and thus result in improved query expressivity.

Chapter 7

Conclusion

This deliverable reports on new methods for achieving scalability, as proposed by work package 2.1, especially on how to do practical approximations and distributed reasoning for ontologies. In the beginning, the focus is, as in the first version of this deliverable, on approximation methods and their applications and results of experiments. In the end we concentrate on distributed reasoning methods and the improvements that they offer.

In the beginning of this deliverable, two major solutions were given to the well known problem of Description Logic reasoners that don't seem to react well for A-Box reasoning when the number of instances of an ontology becomes larger. In Chapter 2 we gave an overview over such two approximation methods and described how they can be applied to the problem of instance retrieval. We also presented the results of experiments done with these methods applied to the above stated problem, using Gene Ontology. The experiments focus on the problem of whether the approximation methods can lead to any reductions in complexity and if the costs for doing this are worthy or not. These experiments and evaluations against benchmarking sets were also conducted with the purpose of improving the instance retrieval methods from the point of view of scalability of such ontologies with a large number of instances.

In Chapter 3, we described the new SCREECH system for approximate A-Box reasoning, which is based on a language weakening transformation of OWL DL into Datalog. The method has polynomial time complexity and is complete but unsound with respect to OWL DL semantics. A performance evaluation showed, however, that SCREECH achieves improvements in run-time while delivering only very few wrong answers.

Starting from the observation that users don't manage to successfully formulate their own queries over extensive amounts of data, in Chapter 4 we described how we developed a specialized rule language which implements query rewriting of RDF rules in order to provide personalized information access to distributed resources (on the semantic web). This approach was created as a personalized search service of a personal learning assistant, which combines user preferences and user query formulation dialogue. By implementing query relaxation and query refinement, the application in this framework has

also been tested on two metadata sets. It proved how RDF queries can be reformulated in order to give good answers, even if the user didn't manage to formulate them accurately.

The architecture of DRAGO, clearly based on a peer-to-peer structure and distributed knowledge management systems, showed in Chapter 5 how instance retrieval and distributed reasoning can be accomplished. It makes use of the distributed contextual reasoning and querying paradigm, which is based among others on semantic mappings and local reasoning. This is a very clear example about how we can benefit from the advantages of distributed reasoning and querying algorithms over (large) OWL ontologies, interrelated by semantic links.

The last contribution in this deliverable described the problem of efficiently querying distributed sources using peer-to-peer structures, especially if collection-wide information is needed. Experiments have been done on a TREC document collection volume consisting of LA Times articles categorized according to the sections they appeared in. They proved that the proposed algorithm scales, even in more volatile networks. Another important improvement is the fact that by adding a separate category index, it allows to cut back on large portions of the network, thus enhancing scalability even more.

Besides the value of the deliverable itself, several solutions to the scalability problem have been presented, specifically in the area of approximate and distributed reasoning for ontologies. Without doubt, we must mention here that these results emerged from fruitful cooperation between different partners, leading to very good ideas. The theoretical solutions were also followed by implementations and thorough test cases to support their novelty and contributions, outlining the effective problem present in the semantic web, that of continuously expanding ontologies and data sets, and the need for improved solutions.

Bibliography

[1] Karl Aberer. P-grid: A self-organizing access structure for p2p information systems. In *In Proceedings of the Sixth International Conference on Cooperative Information Systems (CoopIS)*, Trento, Italy, 2001.

[2] G. Antoniou and F. van Harmelen. *Handbook on Ontologies in Information Systems*, chapter Web Ontology Language: OWL, pages 67–92. Springer Verlag, 2004.

[3] Krzysztof R. Apt, Howard A. Blair, and Adrian Walker. Towards a theory of declarative knowledge. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 89–148. Morgan Kaufmann, Los Altos, CA, 1988.

[4] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P F. Patel-Schneider. *The Description Logic Handbook - Theory, Implementation and Applications*. Cambridge University Press, 2003.

[5] Franz Baader and Tobias Nipkow. *Term rewriting and all that*. Cambridge University Press, New York, NY, USA, 1998.

[6] Wolf-Tilo Balke, Wolfgang Nejdl, Wolf Siberski, and Uwe Thaden. DL meets P2P – distributed document retrieval based on classification and content. In *Proceedings of the 9th European Conference on Research and Advanced Technology for Digital Libraries (ECDL2005)*, 2005.

[7] Wolf-Tilo Balke, Wolfgang Nejld, Wolf Siberski, and Uwe Thaden. Progressive distributed top-k retrieval in peer-to-peer networks. In *Proceedings of the 21st International Conference on Data Engineering (ICDE 2005)*, 2005.

[8] Wolf-Tilo Balke and Matthias Wagner. Through different eyes: assessing multiple conceptual views for querying web services. In Stuart I. Feldman, Mike Uretsky, Marc Najork, and Craig E. Wills, editors, *WWW (Alternate Track Papers & Posters)*, pages 196–205. ACM, 2004.

[9] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D.L. McGuinness, P.F. Patel-Schneider, and L. Andrea Stein. OWL Web Ontology Language Reference. W3C Recommendation, February 2004. `www.w3.org/TR/owl-ref`.

[10] A. Borgida and L. Serafini. Distributed Description Logics: Assimilating Information from Peer Sources. *Journal of Data Semantics*, 1:153–184, 2003.

[11] P. Bouquet, F. Giunchiglia, F. van Harmelen, L. Serafini, and H. Stuckenschmidt. C-OWL: Contextualizing Ontologies. In *Proceedings of the 2d International Semantic Web Conference (ISWC 2003)*, pages 164–179, 2003.

[12] P. Bouquet, F. Giunchiglia, F. van Harmelen, L. Serafini, and H. Stuckenschmidt. Contextualizing Ontologies. *Journal on Web Semantics*, 1(4):325–343, 2004.

[13] P. Bouquet, L. Serafini, and S. Zanobini. Semantic Coordination: A New Approach and an Application. In *Proceedings of the Sencond Internatinal Semantic Web Conference (ISWC 2003)*, volume 2870 of *Lecture Notes in Computer Science*, pages 130–145. Springer Verlag, 2003.

[14] Craig Boutilier, Ronen I. Brafman, Holger H. Hoos, and David Poole. Reasoning with conditional ceteris paribus preference statements. In Kathryn B. Laskey and Henri Prade, editors, *UAI*, pages 71–80. Morgan Kaufmann, 1999.

[15] Ronen I. Brafman, Carmel Domshlak, Solomon E. Shimony, and Yael Silver. Tcp-nets for preferences over sets. In *WS on Advances in Preference Handling*, 2005.

[16] Nicolas Bruno, Luis Gravano, and Amélie Marian. Evaluating top-k queries over web-accessible databases. In *Proceedings of the 18th International Conference on Data EngineeringE*, pages 369–. IEEE Computer Society, 2002.

[17] Marco Cadoli and Francesco Scarcello. Semantical and computational aspects of Horn approximations. *Artificial Intelligence*, 119(1), may 2000.

[18] Stefano Ceri. A declarative approach to active databases. In Forouzan Golshani, editor, *ICDE*, pages 452–456. IEEE Computer Society, 1992.

[19] Francisco Matias Cuenca-Acuna, Christopher Peery, Richard P. Martin, and Thu D. Nguyen. PlanetP: Using Gossiping to Build Content Addressable Peer-to-Peer Information Sharing Communities. In *Twelfth IEEE International Symposium on High Performance Distributed Computing (HPDC-12)*. IEEE Press, June 2003.

[20] Mukesh Dalal. Anytime clausal reasoning. *Annals of Mathematics and Artificial Intelligence*, 22(3–4):297–318, 1998.

[21] Peter Dolog, Nicola Henze, Wolfgang Nejdl, and Michael Sintek. Personalization in distributed e-learning environments. In *Proc. of WWW2004 — The Thirteen International World Wide Web Conference*, New Yourk, May 2004. ACM Press.

[22] Peter Dolog and Michael Schäfer. A framework for browsing, manipulating and maintaining interoperable learner profiles. In Liliana Ardissono, Paul Brna, and

Antonija Mitrović, editors, *Proc. User Modeling 2005: 10th International Conference, UM 2005*, volume 3538 of *LNAI*, Edinburgh, Scotland, UK, July 2005. Springer.

[23] The dublin core metadata initiative. http://dublincore.org/.

[24] M. Ehrig, Ch. Tempich, J. Broekstra, F. van Harmelen, M. Sabou, R. Siebes, S. Staab, and H. Stuckenschmidt. A Metadata Model for Semantics-Based Peer-to-Peer Systems. In *Proceedings of the 2d Konferenz Professionelles Wissensmanagement*, 2003.

[25] Thomas Eiter, Nicola Leone, Christinel Mateis, Gerald Pfeifer, and Francesco Scarcello. A deductive system for nonmonotonic reasoning. In Jürgen Dix and et al, editors, *Proceedings of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'97)*, volume 1265 of *Lecture Notes in Artificial Intelligence*. Springer, Berlin, 1997.

[26] François Fages. Consistency of Clark's completion and existence of stable models. *Journal of Methods of Logic in Computer Science*, 1:51–60, 1994.

[27] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. In *PODS '01: Proceedings of the twentieth ACM SIGMOD- SIGACT-SIGART symposium on Principles of database systems*, pages 102–113, New York, NY, USA, 2001. ACM Press.

[28] Terry Gaasterland, Parke Godfrey, and Jack Minker. An overview of cooperative answering. *Journal of Intelligent Information Systems*, 1(2):123–157, 1992.

[29] Terry Gaasterland, Parke Godfrey, and Jack Minker. Relaxation as a platform for cooperative answering. *Journal of Intelligent Information Systems*, 1(3/4):293–321, 1992.

[30] Terry Gaasterland and Jorge Lobo. Qualified answers that reflect user needs and preferences. In *Proceedings of 20th International Conference on Very Large Data Bases (VLDB94)*, pages 309–320, 1994.

[31] Terry Gaasterland and Jorge Lobo. Qualifying answers according to user needs and preferences. *Fundamenta Informaticae*, 32(2):121–137, 1997.

[32] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

[33] F. Giunchiglia, P. Shvaiko, and M. Yatskevich. Semantic Matching. In *First European Semantic Web Symposium (ESWS 2004)*, pages 61–75, 2004.

[34] B. Cuenca Grau, B. Parsia, and E. Sirin. Pellet: An OWL DL Reasoner. In *Proceedings of the Third International Semantic Web Conference (ISWC 2004)*, 2004. `http://www.mindswap.org/2003/pellet/index.shtml`.

[35] B. Cuenca Grau, B. Parsia, and E. Sirin. QOM: Quick Ontology Mapping. In *Proceedings of the Third International Semantic Web Conference (ISWC 2004)*, 2004.

[36] B. Cuenca Grau, B. Parsia, and E. Sirin. Working with Multiple Ontologies on the Semantic Web. In S. A. McIlraith, D. Plexousakis, and F. van Harmelen, editors, *Proceedings of the Third International Semantic Web Conference (ISWC 2004)*, volume 3298 of *Lecture Notes in Computer Science*, pages 620–634. Springer Verlag, 2004.

[37] P. Groot, A. ten Teije, and F. van Harmelen. Towards a structured analysis of approximate problem solving: a case study in classification. In *Proceedings of the Ninth International Conference on Principles of Knowledge Representation and Reasoning (KR'04)*, Whistler, Colorado, June 2004.

[38] Perry Groot, Heiner Stuckenschmidt, and Holger Wache. Approximating description logic classification for semantic web reasoning. In Asunción Gómez-Pérez and Jérôme Euzenat, editors, *The Semantic Web: Research and Applications, Second European Semantic Web Conference, ESWC 2005, Heraklion, Crete, Greece, May 29 - June 1, 2005, Proceedings*, volume 3532 of *Lecture Notes in Computer Science*, pages 318–332. Springer, 2005.

[39] Object Management Group. OMG unified modelling language specification, version 1.3, March 2000. Available at http://www.omg.org/. Accessed on June 1, 2001.

[40] Ulrich Güntzer, Wolf-Tilo Balke, and Werner Kießling. Optimizing multi-feature queries for image databases. In Amr El Abbadi, Michael L. Brodie, Sharma Chakravarthy, Umeshwar Dayal, Nabil Kamel, Gunter Schlageter, and Kyu-Young Whang, editors, *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt*, pages 419–428. Morgan Kaufmann, 2000.

[41] V. Haarslev and R. Möller. RACE system description. In *Proceedings of the 1999 DL Workshop*, CEUR Electronic Workshop Proceedings, pages 130–132, 1999.

[42] V. Haarslev and R. Möller. High performance reasoning with very large knowledge bases: A practical case study. In *IJCAI'2001*, pages 161–168, 2001.

[43] V. Haarslev and R. Möller. RACER system description. In *IJCAR'2001*, volume 2083 of *LNAI*, pages 701–705. Springer, 2001.

[44] Pat Hayes. Rdf semantics. Recommendation, W3C, 2004.

[45] P. Hitzler. Towards a systematic account of different semantics for logic programs. *Journal of Logic and Computation*, 15(3):391–404, 2005.

[46] Pascal Hitzler and Matthias Knorr. Towards a unified theory of logic programming semantics: Level mapping characterizations of disjunctive stable models. Technical report, AIFB, University of Karlsruhe, 2005. Available from http://www.aifb.uni-karlsruhe.de/WBS/phi.

[47] Pascal Hitzler and Matthias Wendt. A uniform approach to logic programming semantics. *Theory and Practice of Logic Programming*, 5(1–2):123–159, 2005.

[48] I. Horrocks. The FaCT System. In *TABLEAUX'98*, volume 1397 of *LNAI*, pages 307–312. Springer, 1998.

[49] I. Horrocks. Using an Expressive Description Logic: FaCT or Fiction? In *KR'98*, pages 636–647. Morgan Kaufmann, 1998.

[50] I. Horrocks, L. Li, D. Turi, and S. Bechhofer. The InstanceStore: DL Reasoning with Large Numbers of Individuals. In *Proceedings of the International Workshop on Description Logics (DL 2004)*, pages 31–40, 2004.

[51] I. Horrocks and P. F. Patel-Schneider. FaCT and DLP. In *Proceedings of the Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 1998)*, pages 27–30, 1998.

[52] I. Horrocks and P. F. Patel-Schneider. A Proposal for an OWL Rules Language. In *Proc. of the Thirteenth Int'l World Wide Web Conf.(WWW 2004)*. ACM, 2004.

[53] I. Horrocks, U. Sattler, and S. Tobies. A Description Logic with Transitive and Converse Roles, Role Hierarchies and Qualifying Number Restriction. Technical Report 99-08, Technische Universität Dresden, LTCS, 1999.

[54] I. Horrocks, U. Sattler, and S. Tobies. Practical Reasoning for Very Expressive Description Logics. *Logic Journal of the IGPL*, 8(3):239–263, 2000.

[55] I. Horrocks and S. Tessaris. A Conjunctive Query Language for Description Logic Aboxes. In *AAAI*, pages 399–404, 2000.

[56] Ian Horrocks, Lei Li, Daniele Turi, and Sean Bechhofer. The Instance Store: DL reasoning with large numbers of individuals. In *Proceedings of the International Workshop on Description Logics, DL2004, Whistler, Canada*, pages 31–40, 2004.

[57] U. Hustadt, B. Motik, and U. Sattler. Reasoning for Description Logics around $\mathcal{SHIQ}$ in a Resolution Framework. Technical Report 3-8-04/04, FZI, Karlsruhe, Germany, April 2004. http://www.fzi.de/wim/publikationen.php?id=1172.

[58] U. Hustadt, B. Motik, and U. Sattler. Reducing $\mathcal{SHIQ}^-$ Description Logic to Disjunctive Datalog Programs. In *Proc. of the 9th Conference on Knowledge Representation and Reasoning (KR2004)*. AAAI Press, June 2004.

[59] U. Hustadt, B. Motik, and U. Sattler. Data complexity of reasoning in very expressive description logics. In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland*, pages 466–471, 2005.

[60] Ullrich Hustadt, Boris Motik, and Ulrike Sattler. Reasoning in description logics with a concrete domain in the framework of resolution. In Ramon López de Mántaras and Lorenza Saitta, editors, *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain, August 22-27, 2004*, pages 353–357. IOS Press, 2004.

[61] Werner Kießling. Foundations of preferences in database systems. In *VLDB*, pages 311–322, 2002.

[62] Werner Kießling and Gerhard Köstler. Preference sql - design, implementation, experiences. In *Proceedings of 28th International Conference on Very Large Data Bases (VLDB02)*, pages 990–1001, 2002.

[63] R. Korfhage. *Information Storage and Retrieval*. John Wiley, New York, 1997.

[64] M. Lacroix and Pierre Lavency. Preferences; putting more knowledge into queries. In Peter M. Stocker, William Kent, and Peter Hammersley, editors, *Proceedings of 13th International Conference on Very Large Data Bases (VLDB87)*, pages 217–225. Morgan Kaufmann, 1987.

[65] Yuhua Li, Zuhair A. Bandar, and David McLean. An approach for measuring semantic similarity between words using multiple information sources. *IEEE Transactions on Knwoledge and Data Engineering*, 15(4), 2003.

[66] John W. Lloyd. *Foundations of Logic Programming*. Springer, Berlin, 1988.

[67] Jie Lu and Jamie Callan. Federated search of text-based digital libraries in hierarchical peer-to-peer networks. In *European Colloquium on IR Research (ECIR 2005)*, 2005.

[68] C. Lutz. Description Logics with Concrete Domains — A Survey. In *Advances in Modal Logics*, volume 4. King's College Publications, 2003.

[69] A. Maedche, B. Motik, N. Silva, and R. Volz. MAFRA - a Mapping Framework for Distributed Ontologies. In *Proc. of Knowledge Engineering and Knowledge Management (EKAW-02)*, volume 2473 of *Lecture Notes in Computer Science*. Springer, 2002.

[70] David Makinson. *Bridges from Classical to Nonmonotonic Logic*, volume 5 of *Texts in Computing*. King's College Publications, London, 2005.

[71] M.Bonifacio, P.Bouquet, and P.Traverso. Enabling Distributed Knowledge Management. Managerial and Technological Implications. *Novatica and Informatik/Informatique*, III(1), 2002.

[72] Boris Motik, Ulrike Sattler, and Rudi Studer. Query answering for OWL-DL with rules. In *Proceedings of the 3rd International Semantic Web Conference (ISWC2004), Hiroshima, Japan, November 2004*, 2004. To appear.

[73] A. Motro. Flexx: A tolerant and cooperative user interface to database. *IEEE Transactions on Knowledge and Data Engineering*, 2(2):231–245, 1990.

[74] Wolfgang Nejld, Wolf Siberski, Uwe Thaden, and Wolf-Tilo Balke. Top-k query evaluation for schema-based peer-to-peer networks. In *Proceedings of 3rd International Semantic Web Conference (ISWC 2004)*, 2004.

[75] M. Nilsson. Ims metadata rdf binding guide. http://kmr.nada.kth.se/el/ims/metadata.html, May 2001.

[76] N. F. Noy and M. A. Mussen. The PROMPT Suite : Interactive Tools for Ontology Merging and Mapping. *International Journal of Human-Computer Studies*, 56(6):983–1024, 2003.

[77] Assosiation of Computing machinery. The acm computer classification system. http://www.acm.org/class/1998/, 2002.

[78] B. Omelayenko. RDFT: A Mapping Meta-Ontology for Business Integration. In *Proc. of the Workshop on Knowledge Transformation for the Semantic Web at the 15th European Conference on Artificial Intelligence (KTSW2002)*, pages 77–84, 2002.

[79] George Papamarkos, Alexandra Poulovassilis, and Peter T. Wood. Event-condition-action rule languages for the semantic web. In Isabel F. Cruz, Vipul Kashyap, Stefan Decker, and Rainer Eckstein, editors, *SWDB*, pages 309–327, 2003.

[80] N. Paton. *Active Rules in Database Systems*. Springer, 1999.

[81] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content addressable network. In *Proceedings of the 2001 Conference on applications, technologies, architectures, and protocols for computer communications*. ACM Press, 2001.

[82] Grzegorz Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*. World Scientific, 1997.

[83] C. Sakama and K. Inoue. An alternative approach to the semantics of disjunctive logic programs and deductive databases. *Journal of Automated Reasoning*, 13:145–172, 1994.

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

[85] Mario Schlosser, Michael Sintek, Stefan Decker, and Wolfgang Nejdl. HyperCuP—Hypercubes, Ontologies and Efficient Search on P2P Networks. In *International Workshop on Agents and Peer-to-Peer Computing*, Bologna, Italy, July 2002.

[86] B. Selman and H. A. Kautz. Knowledge compilation using Horn approximations. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, pages 904–909, 1991.

[87] L. Serafini, A. Borgida, and A. Tamilin. Aspects of Distributed and Modular Ontology Reasoning. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI 2005)*, 2005.

[88] L. Serafini and F. Roelofsen. Satisfiability for Propositional Contexts. In *Proceedings of the Principles of Knowledge Representation and Reasoning (KR 2004)*, 2004. Accepted for publication.

[89] Wolf Siberski and Uwe Thaden. A simulation framework for schema-based query routing in P2P-networks. In *1st International Workshop on Peer-to-Peer Computing & DataBases(P2P& DB 2004*, 2004.

[90] Patrik Simons, Ilkka Niemelä, and Timo Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1–2):181–234, 2002.

[91] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 Conference on applications, technologies, architectures, and protocols for computer communications*. ACM Press, 2001.

[92] H. Stuckenschmidt, L. Serafini, and H. Wache. Reasoning about Ontology Mappings. Technical report, ITC-IRST, Trento, 2005.

[93] H. Stuckenschmidt and F. van Harmelen. Approximating terminological queries. In *Proceedings of the Fifth International Conference on Flexible Query Answering Systems FQAS 2002*, Lecture Notes in Artificial Intelligence, Copenhagen, Denmark, 2002. Springer Verlag.

[94] Heiner Stuckenschmidt, Anita de Waard, Ravinder Bhogal, Christiaan Fluit, Arjohn Kampman, Jan van Buel, Erik van Mulligen, Jeen Broekstra, Ian Crowlesmith, Frank van Harmelen, and Tony Scerri. Exploring large document repositories with rdf technology - the dope project. *IEEE Intelligent Systems*, 2004. to appear.

[95] C. Tang, Z. Xu, and M. Mahalingam. PeerSearch: Efficient information retrieval in peer-peer networks. Technical Report HPL-2002-198, Hewlett-Packard Labs, 2002.

[96] S. Tessaris. *Questions and Answers: Reasoning and Querying in Description Logic*. PhD thesis, Department of Computer Science, University of Manchester, UK, 2001.

[97] S. Tessaris and I. Horrocks. Abox Satisfiability Reduced to Terminological Reasoning in Expressive Description Logics. In *Proceedings of the Nineth International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR 2002)*, volume 2514 of *Lecture Notes in Computer Science*, pages 435–449. Springer Verlag, 2002.

[98] S. Tobies. The Complexity of Reasoning with Cardinality Restrictions and Nominals in Expressive Description Logics. *Artificial Intelligence Research*, 12:199–217, 2000.

[99] S. Tobies. *Complexity Results and Practical Algorithms for Logics in Knowledge Representation*. PhD thesis, LuFG Theoretical Computer Science, RWTH-Aachen, Germany, 2001.

[100] F. van Harmelen and A. ten Teije. Describing problem solving methods using anytime performance profiles. In *Proceedings of ECAI'00*, pages 181–186, Berlin, August 2000.

[101] Charles L. Viles and James C. French. Dissemination of collection wide information in a distributed information retrieval system. In *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval.*, pages 12–20. ACM Press, 1995.

[102] Charles L. Viles and James C. French. On the update of term weights in dynamic information retrieval systems. In *Proceedings of the 1995 International Conference on Information and Knowledge Management (CIKM)*, pages 167–174. ACM, 1995.

[103] Raphael Volz. *Web Ontology Reasoning with Logic Databases*. PhD thesis, AIFB, University of Karlsruhe, 2004.

[104] W3C. Web ontology language (OWL). www.w3.org/2004/OWL/, 2004.

[105] Holger Wache, Perry Groot, and Heiner Stuckenschmit. Scalable instance retrieval for the semantic web by approximation. In *Proceedings of the 6th International Conference on Web Information Systems Engineering (WISE'05)*, 2005.

[106] Chaokun Wang, Jianzhong Li, and Shengfei Shi. Cell abstract indices for content-based approximate query processing in structured peer-to-peer data systems. In *APWeb*, volume 3007 of *Lecture Notes in Computer Science*. Springer, 2004.

[107] Ian Witten, Alistair Moffat, and Timothy Bell. *Managing Gigabytes*. Morgan Kaufman, Heidelberg, 1999.

[108] Beverly Yang and Hector Garcia-Molina. Designing a super-peer network. In *Proccedings of the 19th International Conference on Data Engineering (ICDE)*, 2003.

Related deliverables

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

Project	Number	**Title** and relationship
KW	D2.1.1	**D2.1.1 Survey of Scalability Techniques for Reasoning with Ontologies** gives an overview of methods for approximating the reasoning.
KW	D2.1.2	**D2.1.2 Methods for Approximate Reasoning** contains earlier reports about investigated approximation methods
KW	D2.1.3.1	**D2.1.3.1 Report on Modularization of Ontologies** contains earlier reports about approaches for distributed reasoning
KW	D2.5.2	**D2.5.2 "Report on Query Language Design and Standardisation"** contains experiments of InstanceStore