NeOn: Lifecycle Support for Networked Ontologies

Integrated Project (IST-2005-027595)

Priority: IST-2004-2.4.7 — "Semantic-based knowledge and content systems"

D1.4.1 Prototypes for Managing Networked Ontologies

Deliverable Co-ordinator: **Yimin Wang, Peter Haase**

Deliverable Co-ordinating Institution: **Universität Karlsruhe – TH (UKARL)**

Other Authors: **Raúl Palma (Universidad Politécnica di Madrid – UPM)**

The goal of this deliverable is to provide applications to support the NeOn networked ontology model to be able to manage networked ontologies in an effective and efficient manner in a distributed networking environment. To achieve this, we provide an integrated architecture for prototypes which consists of Oyster, KAONp2p and KAONWeb systems. The evaluation indicates the prototypes reported in this deliverable are well applicable for managing networked ontologies.

Document Identifier:	NEON/2007/D1.4.1/v1.0	Date due:	February 28, 2007
Class Deliverable:	NEON EU-IST-2005-027595	Submission date:	March 30, 2007
Project start date	March 1, 2006	Version:	v1.0
Project duration:	4 years	State:	Final
		Distribution:	Public

NeOn Consortium

This document is part of a research project funded by the IST Programme of the Commission of the European Communities, grant number IST-2005-027595. The following partners are involved in the project:

Open University (OU) – Coordinator Knowledge Media Institute – KMi Berrill Building, Walton Hall Milton Keynes, MK7 6AA United Kingdom Contact person: Martin Dzbor, Enrico Motta E-mail address: {m.dzbor, e.motta}@open.ac.uk	**Universität Karlsruhe – TH (UKARL)** Institut für Angewandte Informatik und Formale Beschreibungsverfahren – AIFB D-76128 Karlsruhe Germany Contact person: Peter Haase E-mail address: pha@aifb.uni-karlsruhe.de
Universidad Politécnica de Madrid (UPM) Campus de Montegancedo 28660 Boadilla del Monte Spain Contact person: Asunción Gómez Pérez E-mail address: asun@fi.ump.es	**Software AG (SAG)** Uhlandstrasse 12 64297 Darmstadt Germany Contact person: Walter Waterfeld E-mail address: walter.waterfeld@softwareag.com
Intelligent Software Components S.A. (ISOCO) Calle de Pedro de Valdivia 10 28006 Madrid Spain Contact person: Richard Benjamins E-mail adress: rbenjamins@isoco.com	**Institut 'Jožef Stefan' (JSI)** Jamova 39 SL–1000 Ljubljana Slovenia Contact person: Marko Grobelnik E-mail address: marko.grobelnik@ijs.si
Institut National de Recherche en Informatique et en Automatique (INRIA) ZIRST – 665 avenue de l'Europe Montbonnot Saint Martin 38334 Saint-Ismier France Contact person: Jérôme Euzenat	**University of Sheffield (USFD)** Dept. of Computer Science Regent Court 211 Portobello street S14DP Sheffield United Kingdom Contact person: Hamish Cunningham
Universität Kolenz-Landau (UKO-LD) Universitätsstrasse 1 56070 Koblenz Germany Contact person: Steffen Staab E-mail address: staab@uni-koblenz.de	**Consiglio Nazionale delle Ricerche (CNR)** Institute of cognitive sciences and technologies Via S. Marino della Battaglia 44 – 00185 Roma-Lazio Italy Contact person: Aldo Gangemi E-mail address: aldo.gangemi@istc.cnr.it
Ontoprise GmbH. (ONTO) Amalienbadstr. 36 (Raumfabrik 29) 76227 Karlsruhe Germany Contact person: Jürgen Angele E-mail address: angele@ontoprise.de	**Asociación Española de Comercio Electrónico (AECE)** C/Icalde Barnils, Avenida Diagonal 437 08036 Barcelona Spain Contact person: Jose Luis Zimmerman E-mail address: jlzimmerman@fecemd.org
Food and Agriculture Organization of the United Nations (FAO) Viale delle Terme di Caracalla 00100 Rome, Italy Contact person: Marta Iglesias E-mail address: marta.iglesias@fao.org	**Atos Origin S.A. (ATOS)** Calle de Albarracín, 25 28037 Madrid Spain Contact person: Tomás Pariente Lobo E-mail address: tomas.parientelobo@atosorigin.com

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 writing parts of this document:

- Universität Karlsruhe – TH (UKARL)
- Universidad Politécnica di Madrid (UPM)

Change Log

Version	Date	Amended by	Changes
0.1	15-08-2006	Yimin Wang	Creation
0.2	15-10-2006	Yimin Wang	Overview
0.3	10-11-2006	Yimin Wang	KAONp2p
0.4	28-11-2006	Raul Palma	Oyster
0.5	06-12-2006	Yimin Wang	Revising with Peter's comments
0.6	18-12-2006	Yimin Wang	KAONWeb
0.7	29-01-2007	Yimin Wang	Revising and adding conclusions
0.8	06-02-2007	Raul Palma	Updating Oyster instructions
0.9	08-02-2007	Yimin Wang	Version for pre-review
0.95	05-03-2007	Yimin Wang	Addressing the comments from reviewer
1.0	20-03-2007	Peter Haase	QA

Executive Summary

In this deliverable, we present software prototypes for managing networked ontologies that are defined in NeOn Project Deliverable 1.1.1 [HRW+06]. The networked ontology model calls for a new architecture for ontology-based systems, therefore it is necessary to summarize the existing work on distributed applications and make a proposals for new prototypes.

The prototypes for networked ontology management is motivated by the requirements of handling networked ontologies in the distributed networking environment. The requirements are collected from the case study workpackages in NeOn project, i.e., workpackage 7 and 8. Based on the NeOn architecture defined in Workpackages 6, several systems have already been successfully developed by NeOn project partners:

- Oyster system
- KAONp2p system
- KAONWeb system

We first introduce some foundations to support Oyster, KAONp2p and KAONWeb. The detailed technical description describes the features and particular roles of these systems in the overall integrated architecture for networked ontology management. We perform some experimental evaluations to show that our prototypes are applicable in handling the networked ontologies in efficient way. We also conclude our work and discuss the roadmap to indicate the future directions in T1.4 in Workpackage 1. To facilitate the usage of the systems introduced in this deliverable, we provide the software instructions as appendix in the end.

In a nutshell, this deliverable reports the first step to provide comprehensive software prototypes in the NeOn project. We aim to continue our work in T1.4 by following the existing achievements and explore future advances in managing networked ontologies.

Contents

1 Introduction **9**
1.1 Scope . . . 9
1.2 Motivation . . . 10
1.3 State-of-the-art . . . 11
1.4 Overview of the Deliverable . . . 12

2 Overview of Integrated Architecture for Managing Networked Ontologies **13**
2.1 Distributed System Scenario . . . 13
2.2 The Integrated Architecture . . . 14

3 Foundations for Networked Ontology Management **16**
3.1 Preliminaries . . . 16
3.1.1 The Description Logic $\mathcal{SHIN}(\mathbf{D})$ . . . 16
3.1.2 Conjunctive Queries . . . 18
3.2 Metadata for Distributed Ontologies . . . 18
3.2.1 Ontology Metadata Vocabulary - OMV . . . 19
3.2.2 OMV Extension - Peer Metadata . . . 20
3.2.3 Discovery and Selection of Resources . . . 21

4 Components of Integrated Architecture **22**
4.1 Oyster – A Distributed Ontology Metadata Registry . . . 22
4.1.1 Oyster Design . . . 23
4.1.2 Overview Of Oyster . . . 24
4.1.3 Experience . . . 27
4.1.4 Conclusions and Future Work . . . 28
4.2 KOANp2p – A Decentralized Ontology Query Answering System . . . 28
4.2.1 Overview of KAONp2p . . . 28
4.2.2 Mappings in KAONp2p . . . 30
4.2.3 Query Answering in KAONp2p . . . 31
4.2.4 Evaluation . . . 32
4.2.5 Conclusions and Future Work . . . 33
4.3 KAONWeb – An Infrastructure for Optimizing Distributed Query Answering over Networked ontologies . . . 34
4.3.1 Query Answering in KAONWeb . . . 34
4.3.2 Implementation of KAONWeb . . . 39
4.3.3 Evaluation . . . 40

4.3.4 Conclusions and Future Work 42

5 Conclusion **44**
5.1 Summary 44
5.2 Future Roadmap 45

A Appendix: Software User Instructions **46**
A.1 Oyster User Instructions 46
A.1.1 Download and Setup 46
A.1.2 Usage 47
A.2 KOANp2p User Instructions 49
A.2.1 Download and Setup 49
A.2.2 Usage 49
A.3 KAONWeb User Instructions 52
A.3.1 Download and Setup 52
A.3.2 Usage 52

Bibliography **56**

List of Tables

3.1 Translation of $\mathcal{SHIN}(\mathbf{D})$ into FOL 17

List of Figures

1.1 The NeOn Big Picture . . . 10

2.1 Decentralized Structure for human, ontology and human- ontology interactions. This three nodes presentation is a typical abstract subset of the human-system network. . . . 14
2.2 The Integrated Architecture . . . 15

3.1 General OMV overview . . . 19
3.2 Overview of the P-OMV Ontology . . . 20

4.1 Overview of Oyster Architecture . . . 23
4.2 Oyster GUI . . . 26
4.3 Overview of KAONp2p Architecture . . . 29
4.4 Evaluation Results for LUBM 1..8 . . . 33
4.5 Cost of Mappings between Heterogeneous Ontologies . . . 34
4.6 Scenario of hybrid query answering. The user sends the query to the distributed query answering system on node 1 that processes the query, splitting the $\mathcal{IS}$, queries over corresponding ontologies, collects the results and sends them back to the user. . . . 36
4.7 Web Service structure in KAONWeb . . . 40
4.8 Distributed query answering performance *increase* against centralized query answering. . . . 41
4.9 The performance *increase* with respect to the degree of coupling. . . . 42

A.1 Personalizing columns . . . 48
A.2 Oyster import result . . . 49
A.3 KAONp2p runtime snapshot. . . . 50
A.4 Searching and integration for shared resources. . . . 51
A.5 Ontology mappings between different domain ontologies. . . . 52
A.6 Combination of ontologies to extract the relevant resources. . . . 53
A.7 Publishing and reuse component use case example. . . . 54
A.8 Query answering use case example. . . . 55

Chapter 1

Introduction

In this chapter, we discuss the scope of this deliverable, the motivation of our work, the state-of-the-art research topics that are related to this deliverable, and how this deliverable is organized.

1.1 Scope

One goal of the NeOn project is to provide application support for the next generation of the Semantic Web. In the first several years of the Semantic Web research, the applications usually acted as localized systems rather than distributed systems in a networked scenario. In recent years, people began to develop applications by taking into account the requirements of internet-oriented information systems. Server-based applications and their corresponding clients, Peer-to-Peer applications, and other web-based applications have been widely developed or are actively being developed for the semantic systems.

This deliverable is part of the work in Workpackage 1 "Dynamics of Networked Ontologies" of the NeOn project. The goal of this work package is to develop an integrated approach for the evolution process of networked ontologies and related metadata. For this individual part of Workpackage 1, we will develop new prototypes for networked ontology management that supports handling the ontological data with consideration of the complex relationships in the ontology network, such as dependencies, mappings, versions and (in) consistency aspects. Therefore, introducing the prototype for networked ontology management should take all those aspects thoroughly into account.

The networked ontology model has many different features that makes the networked ontology-based applications different from the traditional ontology-based applications. The features of applications that use networked ontologies include:

- *Dynamics and interconnectivity.* On the one hand, the dynamics of data determine that the data are tautological from time to time. On the other hand, because the data sets are interconnected with each other, when one data set is changed, the other data sets that are interconnected with this data set are also required to be changed accordingly. To implement this, mechanisms like monitors and triggers will be taken into consideration.

- *Distributed working scenario.* The data in the networked ontology-based applications is usually distributed rather than centralized, which means, all the distributed nodes may own certain amount of data.

- *Multilingual aspects.* As one important use case for NeOn, applications serving large international organizations, such as FAO, usually work with data from many places around the world that is naturally encoded in many languages.

As we see from Figure 1.1, Workpackage 1 plays the central part of the research and development workpackages in the NeOn project. The tasks of Workpackage 1 are intensively related to other workpackages.

As the central application part of Workpackage 1, Task 1.4 aims to provide the prototypes for networked ontology management to other work package partners, in particular, to Workpackage 6 regarding the NeOn infrastructure and to the case study in Workpackage 7 about constructing and over-fishing alert system and fishery ontology life-cycle management system.

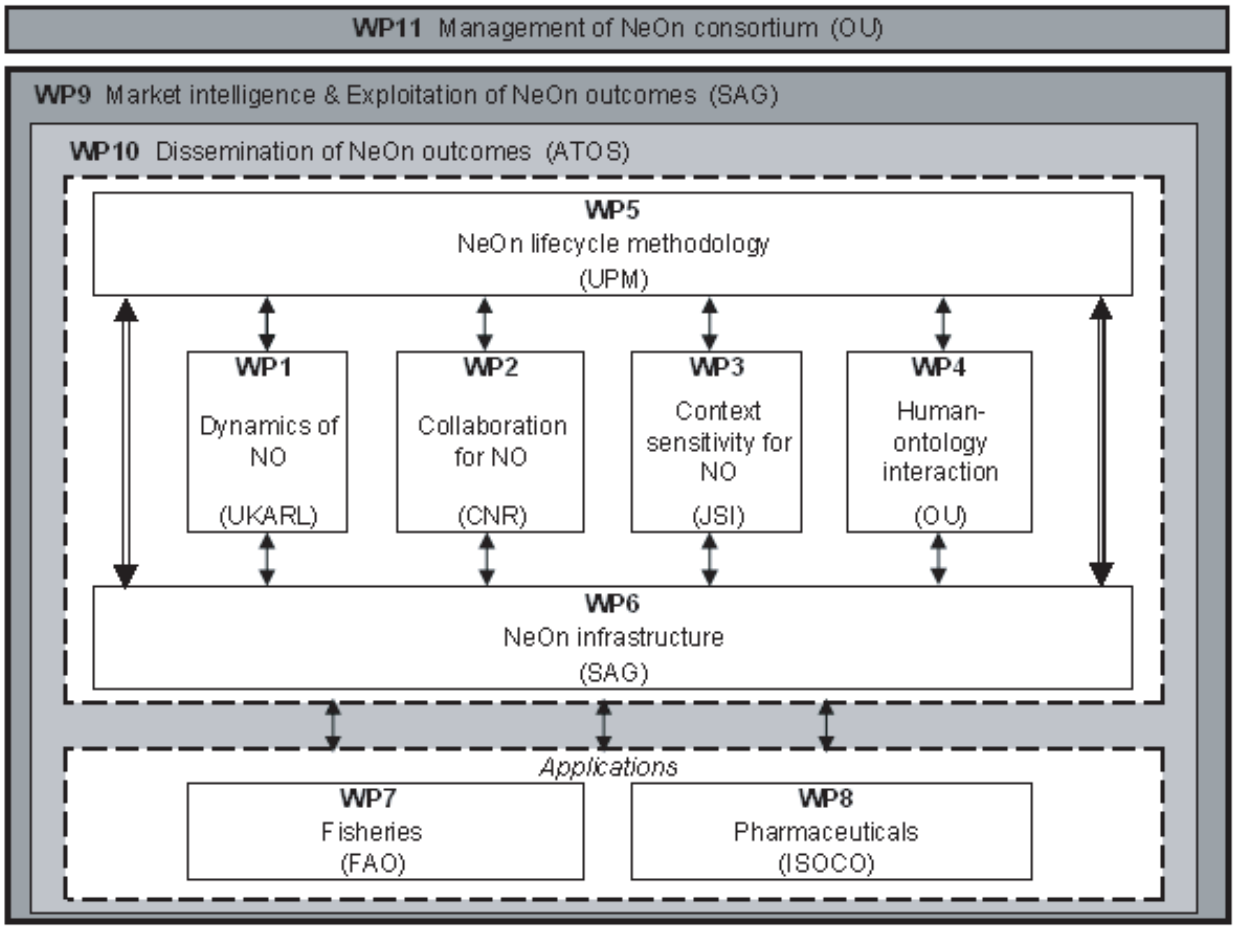

Figure 1.1: The NeOn Big Picture

1.2 Motivation

The realization of real-life applications in the Semantic Web requires the ability to deal with heterogeneous ontologies fragmented and distributed over multiple autonomous nodes. In recent years much progress has been made in providing efficient and scalable reasoning support over expressive ontologies: We now have a number of reasoners such as FaCT++ [TH04], RacerPro (`http://www.racer-systems.com/`), Pellet [SP04], KAON2 (`http://kaon2.semanticweb.org/`) and OntoBroker (`http://www.ontoprise.de/`) that allow to handle ontologies with reasonable size and complexity. However, these reasoners typically assume centralized and closed settings, where a number of known ontologies are integrated in one local node. Moreover, ontology re-use is rather difficult within a centralized setting. It is difficult to find and share ontologies available among the community, because standard metadata information for documenting and annotating ontologies is not widely being used. This leads to the problem of having many isolated ontologies created by many different parties that causes duplicated efforts. Besides the costs of the duplicated efforts, this also hampers interoperability between ontology-based applications. Also, in a centralized setting, it is often more difficult to optimize the overall performance of the system than decentralized setting, which is also one of the

arising challenges in the Semantic Web research.

The decentralized approach provides an ideal solution for users that require a registry to which they have full access and can perform any operation (e.g. reasoning, register, update or delete metadata) without any consequences to other users, similar to managing their personal favorite song list. For example, users from academia or industry might use a personal registry for a task dependant investigation or ontology engineers, might use it during their ontology development process to query over different ontologies and capture information about different ontology versions. Therefore, we argue that the decentralized infrastructure better reflects the spirit of the open Semantic Web, where ontologies are distributed over a number of autonomous nodes. This decentralized infrastructure also features that in each decentralized peer, there is a server that serves remote peers with their particular functionalities. This feature provides a potential opportunity to optimize the overall performance of the decentralized system by appropriately distributing the tasks in the network.

When dealing with such decentralized infrastructures in the distributed scenario, we need to consider a number of arising challenges that are currently not addressed in centralized systems. The first fundamental challenge is the coordination of autonomous nodes, i.e. the ability to manage the organization of the interaction between nodes. In the case of completely centralized architectures, one node has complete control over all other nodes, whereas in completely decentralized architectures there is no central control and all nodes act autonomously. A particularly important coordination task is that of discovering and selecting resources relevant for answering queries as well as routing of requests: How do you find the right nodes that are able to answer a given query in a decentralized system in a scalable manner without any centralized servers or hierarchy? Related to the problem of autonomy is that of heterogeneity: To enable interoperability between nodes in large distributed information systems based on heterogeneous ontologies, it is necessary to specify how the ontologies residing at a particular node correspond to ontologies residing at other nodes. It is also necessary to formally define the notion of a mapping between ontologies. Finally, we need reasoning algorithms that can efficiently deal with ontologies distributed over multiple nodes, taking into account the semantics of the mappings between the individual ontologies. For efficiency reasons, it is important to devise methods that do not require the integration of ontologies in a single node, but that only retrieve the information relevant for the particular reasoning task.

1.3 State-of-the-art

There are several threads of research work that are related to this deliverable: (1) The use of semantics and metadata and corresponding registries for the coordination and organization of distributed information systems, (2) representation of mappings for ontology integration, and (3) approaches to distributed reasoning.

The use of metadata for addressing coordination problems has a history in different communities. In the past, there have been various proposals for modeling metadata of ontologies. Unfortunately, none of them has been accepted as a standard, some proposals, such as Dublin Core, were too general, others were limited in applicability. [MMS+03] has proposed an ontology meta-ontology (OMO) for a distributed ontology registry. The focus of the registry is on locating, re-using and evolving existing ontologies rather than supporting particular reasoning tasks. Semantic representation of resources have further been successfully applied to organize distributed systems with *semantic overlay networks*. In these semantic overlay networks, links are created according to semantic relationships between the nodes. The neighborhood thus mirrors semantic relationships between the peers. For example, Gridvine [ACMHP04] uses the semantic overlay for managing and mapping data and metadata schemas, on top of a physical layer consisting of a structured Peer-to-Peer overlay network called P-Grid. A similar approach is taken in our infrastructure, where nodes maintain a registry with metadata about acquainted nodes, which is used to select relevant nodes and ontologies for answering a given query. Similarly, we can find several approaches for ontology registries. Many of these systems (Knowledge Zone, DAML ontology library, SchemaWeb Directory, WebOnto, Ontolingua, SHOE) provide ontology storage, searching and management features but in general they are either limited in scope or difficult to use. Usually, they store a limited subset of ontology metadata in order to provide some basic

searching capabilities, similar to the Semantic Web search engine SWOOGLE. Moreover, most of the existing approaches are centralised systems or stand alone applications that are difficult to integrate within other systems/applications. Our ontology metadata registry, Oyster, is a distributed system that implements a complete vocabulary to describe ontologies allowing more advanced semantic queries, and that can be easily integrated within another applications.

The task of *ontology integration* using mappings is very related to that of data integration in databases. In [Len02] the author introduces a general framework for data integration and compares existing approaches to data integration (GAV, LAV) along this framework. The work on data integration has been extended and re-applied to ontology integration in [CDL01]. Here the authors follow the classical distinction between LAV and GAV approaches and outline query answering algorithms for these specific settings. In contrast to this work, query answering in our ontology integration system is not bound to these restricted forms of mappings. Further, in data integration the languages to describe the sources and targets are typically very restricted (e.g. express the schemas as plain relations).

With respect to query processing over distributed data, there is a large amount of related work in the area of Peer-to-Peer databases, such as [TIM$^+$03, FKLZ04]. However, most of the work in Peer-to-Peer databases assumes that queries can be answered by simply forwarding the query to other nodes and aggregate the answers afterwards. Such an approach is not sufficient for distributed ontologies. Consider the simple example of two knowledge bases $K_1 = \{\mathsf{Student} \sqsubseteq \mathsf{Person}\}$ and $K_2 = \{\mathsf{Student}(paul)\}$. Evaluating the query $Q(x) := \mathsf{Person}(x)$ against either K_1 or K_2 will return no result; only the combination of the knowledge of K_1 and K_2 will return the desired result. If we simply forward the query to the two KBs separately, then apparently there will be not result returned. In description logics terminology, Peer-to-Peer databases only allow to deal with distributed A-Boxes. On the other hand, reasoners for distributed description logics such as Drago [ST05] currently provides no support for handling assertional knowledge at all.

In our approach, KAONp2p globally integrates the ontology schemata to guarantee the soundness and completeness of query answering over distributed ontologies. On the other hand, KAONWeb, an alternative query answering engine for distributed ontologies, optimizes the query processing by identifying different characteristics of ontologies to seek for possible optimizations. In other words, KAONWeb aims to find a balance between the direct query distribution owned by classic Peer- to-Peer databases and the overall integration of KAONp2p. The details of the similarities and differences of KAONp2p and KAONWeb can be found in Chapter 3.

1.4 Overview of the Deliverable

In Chapter 2, we give an overview for the proposed integrated architecture that applies the networked ontology model and its general prototype by first providing a usage scenario for this prototype. We introduce some foundations for managing networked ontologies in Chapter 3 and afterwards, in Chapter 4, we introduce the components that constitute the prototypes for networked ontology management. The components include metadata for distributed ontologies, Oyster distributed ontology registry, KAONp2p decentralized query answering infrastructure and KAONWeb – optimization for distributed query answering over networked ontologies as an alternative query answering approach with lightweight user interface. KAONp2p and KAONWeb are targeted towards handling different types of distributed data: KAONp2p focuses on the data that are distributively generated with higher autonomy, whereas KAONWeb mainly concerns data that are centrally generated and controlled, but are processed in a distributed manner. Finally, in Chapter 5, we provide a summary and an outlook to subsequent deliverables.

Chapter 2

Overview of Integrated Architecture for Managing Networked Ontologies

2.1 Distributed System Scenario

The distributed system scenario for networked ontology management consists of three kinds of interactions: (1) Human interaction, (2) system interaction and (3) human-system interaction. The development of networked ontology management infrastructure is particularly driven by the two latter aspects.

As shown in Figure 2.1, three kinds of interactions are presented topologically and described in detail as follows:

1. *Human interaction* is one of the most common ways of knowledge exchange, including speaking, writing and so on. It can be either direct (e.g. face-to-face) or indirect (e.g. e-mail), etc. However, this is obviously not the concern of this deliverable – we will focus on the following two aspects, especially the third one.

2. *Human-system interaction* is important to an effective networked ontology management. The user's activities include a series of tasks for managing the ontologies in graphical mode with an intuitive and straightforward interface. The creation and usage of metadata for semantic queries construction in discovering specific ontology is another important topic. Otherwise, the query input and output interface is also a key issue here. This is the central concern of Workpackage 4.

3. *System interaction* is hidden from the user. Typical interactions between system and ontologies basically happen when managing of both local and remote ontologies, mappings between ontologies and propagation of corresponding changing events, which keep ontologies updated together with other ontologies in the network. The second category of system interaction is about ontology metadata management, including also change propagation between metadata and its related ontologies. Moreover, query routing, distribution and answering is also a very important issue in system interaction, especially in this networked scenario with multiple servers and clients.

In the networked scenarios, the aforementioned three scenarios are essential for the deployment of distributed systems. Although in this deliverable, our central concern is the third point as the background infrastructure for the prototypes, we also need to taking human-system interactions into account while developing the prototypes for networked ontology management.

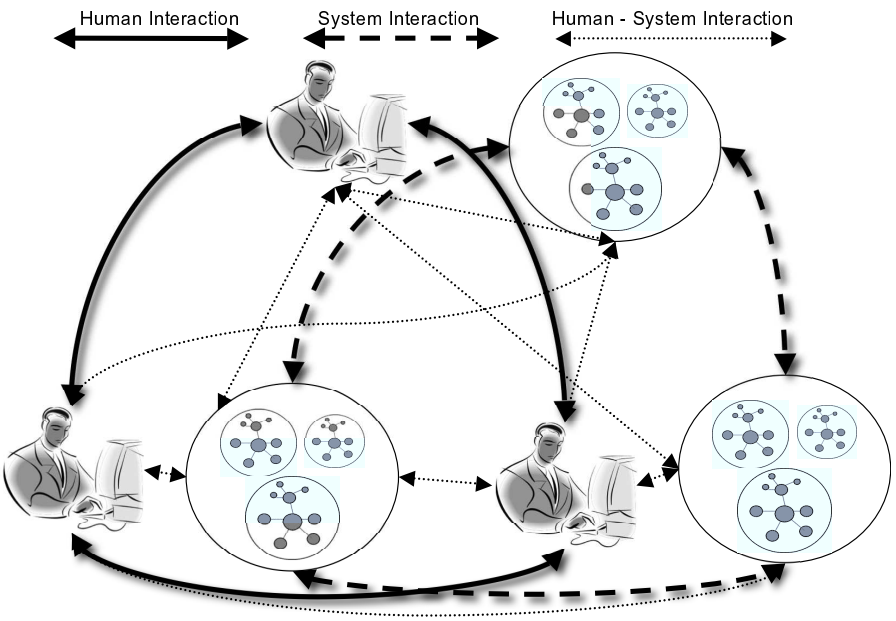

Figure 2.1: Decentralized Structure for human, ontology and human- ontology interactions. This three nodes presentation is a typical abstract subset of the human-system network.

2.2 The Integrated Architecture

The integrated architecture of NeOn aims to provide a unified approach to manage networked ontologies in the complex networking environment. There are three levels of components in this proposed integrated architecture:

1. The integrated infrastructure for low level ontology management.
2. The integrated engineering components that can be tightly or loosely coupled.
3. The integrated GUI components that can be thin or rich.

Figure 2.2 depicts these three levels of components in this integrated architecture, which is depicted by aligning to the NeOn architecture provided in the NeOn project Deliverable 6.2.1.

In this deliverable, the architecture is taken from Workpackage 6, whereas we develop new technologies for managing networked ontologies in the distributed scenario and embed the new developed software to the general NeOn architecture. To be specific, we can see from Figure 2.2:

- The low level ontology management infrastructure consists of three units: (1) Oyster ontology registry services, (2) KAONp2p/KAONWeb distributed reasoner (i.e., KAONWeb is an alternative reasoner that is particularly optimized for distributed computing with a web-based user interfrace.) (3) the Oyster and KOANp2p/KAONWeb rely on the KAON2 ontology reasoner, whereas it is also possible to use Ontobroker as the ontology reasoner and repository service as an alternative.
- The middle level are the integrated engineering components as the background control system for ontology engineering, which can be shared both by the different GUIs. The loosely coupled services in the networked scenario include the Oyster and KAONWeb distributed ontology management system, as well as the collaborative aspect provided by KAONWeb. The tightly coupled services consist of the browsing, mapping and versioning units that are constituted KAONp2p, FOAM and Oyster, respectively. The coordination amongst different services in this level is established by using distributed ontology metadata.

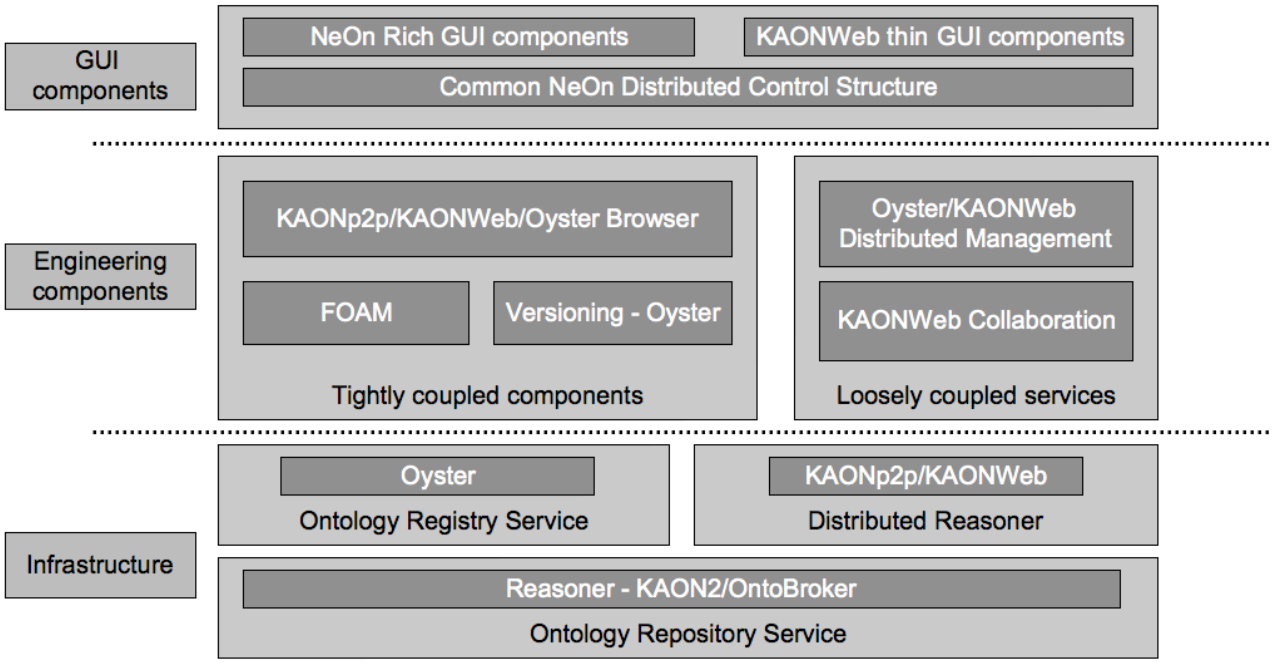

Figure 2.2: The Integrated Architecture

- The upper level GUI components can be divided into two categories – the thin and rich GUI components. In this integrated architecture, the instances of rich GUI components are FOAM, Oyster and KAONp2p. On the other side, KAONWeb system provides thin GUI components based on web application. These GUI components commonly share a common NeOn distributed control structure that is established by a couple of APIs.

In Chapter 4, we introduce the individual components which are essential for this integrated infrastructure. Mature application, such as FOAM, has been developed and can be used directly to this integrated architecture as a well modularized components that does not require complex integration. A detailed introduction of Ontostudio can be found in `http://www.ontoprise.de/content/e1171/e1249/index_eng.html` and the NeOn Workpackage 6 documents. Further technical details of FOAM can be found via the link: `http://www.aifb.uni-karlsruhe.de/WBS/meh/foam/` and [ES05].

Chapter 3

Foundations for Networked Ontology Management

In this chapter, we introduce some foundations for networked ontology management, namely, some prerequisite knowledge to understanding the rest of the deliverable and the distributed ontology metadata as basis for maintaining the relevant information about the networked ontologies.

3.1 Preliminaries

This chapter provides the prerequisite knowledge for understanding this deliverable. In our work, we use the description logic $\mathcal{SHIN}(\mathbf{D})$, and this corresponds to OWL DL with only omission of enumeration as a concept constructor. as our data model and a subset of SPARQL (`http://www.w3.org/TR/rdf-sparql-query/`), namely conjunctive queries in order to query data.

3.1.1 The Description Logic $\mathcal{SHIN}(\mathbf{D})$

Let N_C be a set of *concept names*, N_{R_a} and N_{R_c} sets of *abstract and concrete role names*, respectively, and N_{I_a} and N_{I_c} sets of *abstract and concrete individuals*, respectively. An *abstract role* is an abstract role name or the inverse S^- of an abstract role name S (concrete roles do not have inverses). Finally, let $\mathbf{D}$ be an admissible concrete domain. An *RBox* $KB_{\mathcal{R}}$ is a finite set of transitivity axioms $\mathsf{Trans}(R)$, and role inclusion axioms of the form $R \sqsubseteq S$ and $T \sqsubseteq U$, where R and S are abstract roles, and T and U are concrete roles. The reflexive-transitive closure of the role inclusion relationship is denoted with $\sqsubseteq^*$. A role not having transitive subroles (w.r.t. $\sqsubseteq^*$, for a full definition see [HPS99]) is called a *simple* role.

The set of $\mathcal{SHIN}(\mathbf{D})$ *concepts* is defined by the following syntactic rules, where $A \in \mathbb{N}_C$ 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, c_i are concrete individuals, respectively, and n is a non-negative integer:

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

A *TBox* $KB_{\mathcal{T}}$ is a finite set of concept inclusion axioms $C \sqsubseteq D$, for C and D concepts; an *ABox* $KB_{\mathcal{A}}$ is a finite set of concept and role assertions and individual (in)equalities $C(a)$, $R(a, b)$, $a \approx b$ and $a \not\approx b$, respectively. A $\mathcal{SHIN}(\mathbf{D})$ *knowledge base* KB is a triple $(KB_{\mathcal{T}}, KB_{\mathcal{R}}, KB_{\mathcal{A}})$. The semantics of $\mathcal{SHIN}(\mathbf{D})$ KB can be defined in a direct model-theoretic way to [HPS04] or by a mapping π that translates KB into a first-order formula as specified in Table 3.1.

Table 3.1: Translation of $\mathcal{SHIN}(\mathbf{D})$ into FOL

Mapping Concepts to FOL	
$\pi_y(\top, X)=\top$	$\pi_y(\bot, X)=\bot$
$\pi_y(A, X)=A(X)$	$\pi_y(\neg C, X)=\neg\pi_y(C, X)$
$\pi_y(C \sqcap D, X)=\pi_y(C, X) \wedge \pi_y(D, X)$	$\pi_y(C \sqcup D, X)=\pi_y(C, X) \vee \pi_y(D, X)$
$\pi_y(\forall R.C, X)=\forall y : R(X, y) \rightarrow \pi_x(C, y)$	$\pi_y(\exists R.C, X)=\exists y : R(X, y) \wedge \pi_x(C, y)$
$\pi_y(d, X_1, \ldots, X_m)=$	$d(X_1, \ldots, X_m)$
$\pi_y(\leq n\, R.C, X)=$	$\forall y_1, \ldots, y_{n+1} : \bigwedge R(X, y_i) \wedge \bigwedge \pi_x(C, y_i) \rightarrow \bigvee y_i \approx y_j$
$\pi_y(\geq n\, R.C, X)=$	$\exists y_1, \ldots, y_n : \bigwedge R(X, y_i) \wedge \bigwedge \pi_x(C, y_i) \wedge \bigwedge y_i \not\approx y_j$
$\pi_y(\forall T_1, \ldots, T_m.D, X)=$	$\forall y_1^{\mathsf{c}}, \ldots, y_m^{\mathsf{c}} : \bigwedge T_i(X, y_i^{\mathsf{c}}) \rightarrow \pi_x(D, y_1^{\mathsf{c}}, \ldots, y_m^{\mathsf{c}})$
$\pi_y(\exists T_1, \ldots, T_m.D, X)=$	$\exists y_1^{\mathsf{c}}, \ldots, y_m^{\mathsf{c}} : \bigwedge T_i(X, y_i^{\mathsf{c}}) \wedge \pi_x(D, y_1^{\mathsf{c}}, \ldots, y_m^{\mathsf{c}})$
$\pi_y(\leq n\, T, X)=$	$\forall y_1^{\mathsf{c}}, \ldots, y_{n+1}^{\mathsf{c}} : \bigwedge T(X, y_i^{\mathsf{c}}) \rightarrow \bigvee y_i^{\mathsf{c}} \approx_{\mathbf{D}} y_j^{\mathsf{c}}$
$\pi_y(\geq n\, T, X)=$	$\exists y_1^{\mathsf{c}}, \ldots, y_n^{\mathsf{c}} : \bigwedge T(X, y_i^{\mathsf{c}}) \wedge \bigwedge y_i^{\mathsf{c}} \not\approx_{\mathbf{D}} y_j^{\mathsf{c}}$
Mapping Axioms and KB to FOL	
$\pi(C \sqsubseteq D)=$	$\forall x : \pi_y(C, x) \rightarrow \pi_y(D, x)$
$\pi(R \sqsubseteq S)=$	$\forall x, y : R(x, y) \rightarrow S(x, y)$
$\pi(\mathsf{Trans}(R))=$	$\forall x, y, z : R(x, y) \wedge R(y, z) \rightarrow R(x, z)$
$\pi(C(a))=$	$\pi_y(C, a)$
$\pi(R(a, b))=$	$R(a, b)$
$\pi(a^{(\mathsf{c})} \circ b^{(\mathsf{c})})=$	$a \circ_{(\mathbf{D})} b$ for $\circ \in \{\approx, \not\approx\}$
$\pi(KB)=$	$\bigwedge_{R \in N_R} \forall x, y : R(x, y) \leftrightarrow R^-(y, x) \wedge \bigwedge_{\alpha \in KB_{\mathcal{R}} \cup KB_{\mathcal{T}} \cup KB_{\mathcal{A}}} \pi(\alpha)$

X is a meta variable and is substituted with the actual variable. π_x is obtained from π_y by simultaneously substituting all $y_{(i)}$ with $x_{(i)}$ and π_y with π_x, and vice versa.

3.1.2 Conjunctive Queries

A subset of SPARQL allows formulating conjunctive queries. We use this subset to query $\mathcal{SHIN}(\mathbf{D})$ knowledge bases.

Definition 1 (Conjunctive Queries) *Let KB be a $\mathcal{SHIN}(\mathbf{D})$ knowledge base, and let N_P be a set of predicate symbols, such that all $\mathcal{SHIN}(\mathbf{D})$ concepts and all abstract and concrete roles are in N_P. An* atom *has the form $P(s_1, \ldots, s_n)$, often denoted as $P(\mathbf{s})$, where $P \in N_P$, and s_i are either variables or individuals from KB. An atom is called a* DL-atom *if P is a $\mathcal{SHIN}(\mathbf{D})$ concept, or an abstract or a concrete role; it is called* non-DL-atom *otherwise.*

Let $x_1, \ldots, x_n$ and $y_1, \ldots, y_m$ be sets of distinguished *and* non-distinguished *variables, denoted as $\mathbf{x}$ and $\mathbf{y}$, respectively. A* conjunctive query *over KB, written as $Q(\mathbf{x}, \mathbf{y})$, is a conjunction of atoms $\bigwedge P_i(\mathbf{s_i})$, where the variables in $\mathbf{s_i}$ are contained in either $\mathbf{x}$ or $\mathbf{y}$. Following the semantics of $\mathcal{SHIN}(\mathbf{D})$, we extend the operator π from [MSS04] to translate $Q(\mathbf{x}, \mathbf{y})$ into a first-order formula with free variables $\mathbf{x}$ as follows:*

$$\pi(Q(\mathbf{x}, \mathbf{y})) = \exists \mathbf{y} : \bigwedge \pi(P_i(\mathbf{s_i}))$$

For $Q_1(\mathbf{x}, \mathbf{y_1})$ and $Q_2(\mathbf{x}, \mathbf{y_2})$ conjunctive queries, a query containment *axiom $Q_2(\mathbf{x}, \mathbf{y_2}) \sqsubseteq Q_1(\mathbf{x}, \mathbf{y_1})$ has the following semantics:*

$$\pi(Q_2(\mathbf{x}, \mathbf{y_2}) \sqsubseteq Q_1(\mathbf{x}, \mathbf{y_1})) = \forall \mathbf{x} : \pi(Q_1(\mathbf{x}, \mathbf{y_1})) \leftarrow \pi(Q_2(\mathbf{x}, \mathbf{y_2}))$$

The main inference for conjunctive queries is query answering:

Definition 2 *An* answer *of a conjunctive query $Q(\mathbf{x}, \mathbf{y})$ w.r.t. KB is an assignment θ of individuals to distinguished variables, such that $\pi(KB) \models \pi(Q(\mathbf{x}\theta, \mathbf{y}))$. We use* $\mathsf{Ans}(Q, KB)$ *as a shorthand for the query answer.*

We use SPARQL as the query language for conjunctive query answering over networked ontologies in this deliverable.

Example 1 *The translation of the following SPARQL query*

```
SELECT ?x ?y WHERE { ?x rdf:type lubm:AssistantProfessor .
       ?y rdf:type lubm:Publication . ?y lubm:publicationAuthor ?x}
```

to a conjunctive query is straightforward and results in

$$Q(\mathbf{x}, \mathbf{y}) = \exists \mathbf{x}, \mathbf{y} : \mathsf{AssistantProfessor}(\mathbf{x}) \wedge \mathsf{Publication}(\mathbf{y}) \wedge \mathsf{publicatonAuthor}(\mathbf{x}, \mathbf{y})$$

where $N_P = (\mathsf{AssistantProfessor}, \mathsf{Publication}, \mathsf{publicatonAuthor})$.

3.2 Metadata for Distributed Ontologies

In this section we present a brief overview of our ontology for the representation of metadata about nodes in the network and the resources they provide, i.e. the ontologies. We also show how the task of resource selection is realized using this metadata ontology. In this approach, the nodes advertise descriptions of their resources and can thus establish acquaintances with other nodes. Acquainted nodes can then share data and coordinate their interaction.

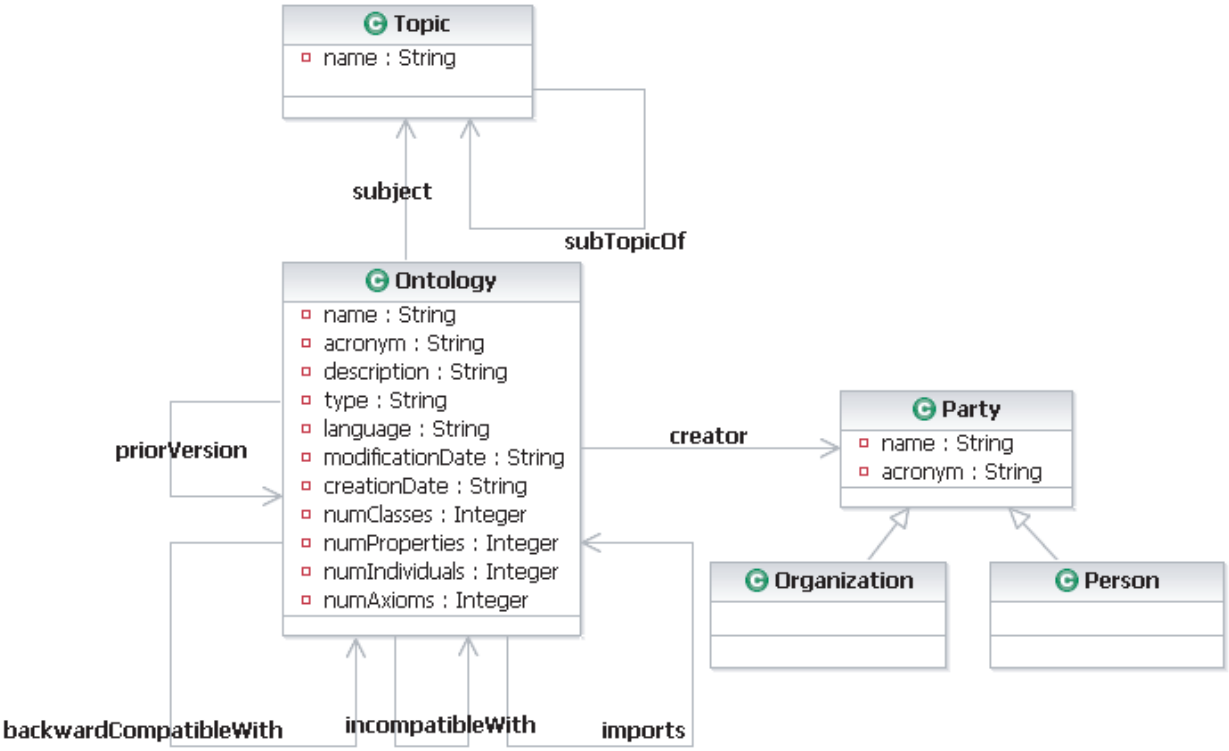

Figure 3.1: General OMV overview

3.2.1 Ontology Metadata Vocabulary - OMV

For the description of ontology metadata we rely on OMV, the Ontology Metadata Vocabulary [HSH+05]. Based on [HRW+06]. let's first briefly recall the structure of OMV and then focus on the properties relevant for the problem of resource discovery and selection. For a complete reference and the complete ontology, we refer the reader to `http://omv.ontoware.org/`. The main classes and properties of the OMV ontology are illustrated in Figure 3.1. We model various types of metadata of ontologies, which we can classify as *Descriptive metadata*, *Provenance metadata* about the creation process, *Dependency metadata* managing relationships with other ontologies such as compatibility, and *Statistical metadata*, e.g. about the size of the ontology in terms of ontology elements, axioms etc.

Descriptive metadata is the most important type of metadata for the discovery and selection of ontologies. Descriptive metadata relates to the domain modeled in the ontology in form of keywords, topic classifications, textual descriptions of the ontology contents etc. It includes the `name` by which the ontology is known, the `language`, its `type` (e.g. *top-level*, *core*, *task*, *domain*, and *application ontology*), and the `subject`: The subject of an ontology provides a classification in terms of the domain. The subject is expressed as a classification against established topic hierarchies, such as the general purpose topic hierarchy DMOZ (`http://dmoz.org/`) or the domain specific ACM topic hierarchy (`http://www.acm.org/class/`) for the computer science domain.

The topics themselves may be organized in a topic ontology organized with relations such as `subTopicOf`.

Provenance metadata provides information about the entities contributing to the creation of the ontology, as well as information about changes since its creation that are significant for its authenticity, integrity and interpretation. It includes e.g. the properties `creator`, i.e. the entity primarily responsible for producing the content of the ontology, the `creationDate` and `modificationDate` indicate when the ontology was first created and modified.

Dependency metadata provides information to support managing relationships with other ontologies.

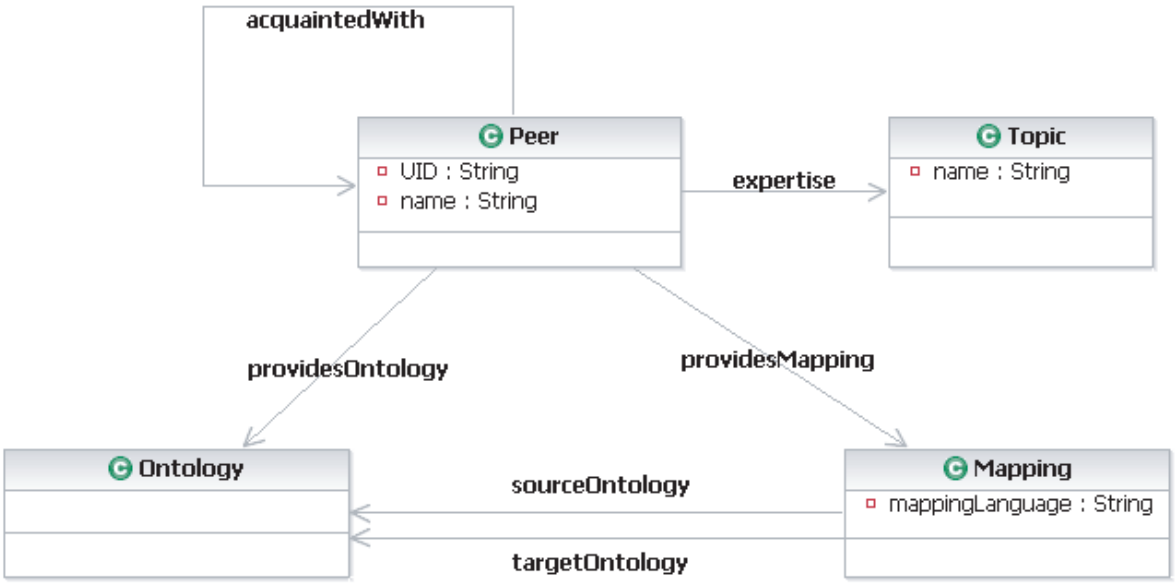

Figure 3.2: Overview of the P-OMV Ontology

These include in particular the `imports`, `backwardCompatibleWith` relation, `incompatibleWith`, `priorVersion`, which identifies the specified ontology as a prior version of the containing ontology.

Statistical metadata relates to the size and type of the ontology. In particular we mention the number of specific ontological primitives (number of classes, properties, individuals, axioms). The availability of such metadata is important for assessing the quantity of information provided, as well as the type of information (e.g. large ABoxes vs. TBoxes), which can for example serve as rough indicators of the costs of processing the ontology.

3.2.2 OMV Extension - Peer Metadata

Besides the ontologies themselves, the second important resources to describe are the nodes managing and providing these informational resources, which we call *peers* in our metadata ontology. The extensions required to model metadata of peers are realized as an extension to the OMV ontology, called P-OMV. Figure 3.2 shows an overview of the P-OMV ontology. The metadata required to describe peers include descriptive information about the peers themselves, their relationship with other peers, as well as information about the resources they provide:

Each peer carries a unique ID (`UID`) to be identified. Depending on the underlying communication infrastructure of the network sublayer, different addressing schemes may be applied. In our implementation of KAONp2p, we simply use IP addresses. In addition to the unique identifier, each peer carries a `name` for identification, which is primarily used for human interpretation. The `expertise` is an abstract description of the peer in terms of some topic ontology. Depending on the application scenario, the expertise of the peer can be the subjects of the ontologies that the peer provides, or a more generic description of expertise.

The property `acquaintedWith` describes the acquaintances of a peer with other peers. The Peer-to-Peer network then consists of local peers, each with a set of acquaintances, which define the Peer-to-Peer network topology. The property `providesOntology` describes the relationship between the peer and the ontologies provided by the peer. It is essential for locating relevant information resources in the network.

The property `providesMapping` is used to describe which mappings between ontologies a peer provides. Mappings are used to describe the correspondences between different ontologies provided by the peers. The

properties `sourceOntology` and `targetOntology` specify the ontologies that are being mapped. In general, mappings need not be symmetric, a distinction between mapping source and target is therefore required. The property `mappingLanguage` is used to indicate the language that is used to express the mapping. In KAONp2p we rely on the formalism for ontology mappings presented in Section 4.2.2, which can be expressed in SWRL. However, other languages may be used for the representation of ontology mappings.

3.2.3 Discovery and Selection of Resources

In our approach to resource discovery and selection we follow the successful approach of expertise-based peer selection [HSvH04], which has already been applied in the Peer-to-Peer systems Bibster [HBE+04] and Oyster [PH05a]. In this approach, peers advertise their resource descriptions according to the metadata ontology in the network to form acquaintances, whereby the peers are fully autonomous in choosing their acquaintances. Moreover, we assume there is no global control in the form of a global registry to manage acquaintances. Acquaintances are managed in a decentralized manner, i.e. by the individual peer using its metadata registry.

For the selection of resources we offer an automated selection of resources based on matching the subject of a query and the expertise according to their semantic similarity, which serves as an indicator for relevance. A subject is an abstraction of a given query expressed in a set of terms from the metadata ontology. The subject can be seen as a complement to an expertise description, as it specifies the required expertise to answer the query. We rely on the notion of a similarity function as defined in [EHSH05]. The similarity function determines the semantic similarity between a subject and an expertise description. As such, an increasing value indicates increasing similarity and relevance. The resource selection algorithm returns a ranked set of resources, where the rank value is equal to the similarity value provided by the similarity function. For example, to answer a query about the subject of *Databases*, the resource selection might identify a peer who provides an ontologies about the subject of the ACM topic is *Information Systems / Database Management* as relevant. For the details of the selection process, we refer the reader to [HSvH04].

Chapter 4

Components of Integrated Architecture

There are three major components in our integrated architecture for managing networked ontologies:

- Oyster, a distributed ontology metadata registry, manages ontologies by using ontology metadata;
- KAONp2p, a decentralized query answering infrastructure, provides the central technical foundation for query answering over distributed ontologies;
- KAONWeb, a web-based distributed query answering system, is an different approach for query distributed ontologies that is particularly optimized for the distributed scenario with a lightweight user interface.

KAONp2p and KAONWeb are similar in that they provide query answering over distributed ontologies, while they address different use cases. It is commonly known that Peer-to-Peer system aims to provide resource sharing as the major scope, especially when resources are created in a decentralized networking environment – that's exactly what KAONp2p is designed for. On the other hand, KAONWeb is developed to support the distributed scenario where the data are created centrally at first and distributed afterwards, therefore the central task for KAONWeb is to provide efficient infrastructure to facilitate distributed query answering.

These components work in an integrated manner have been depicted in Section 2.2 and in this chapter we introduce them individually to discover their functionalities in handling networked ontologies in the distributed environment.

4.1 Oyster – A Distributed Ontology Metadata Registry

Oyster is a Peer-to-Peer application that exploits semantic web techniques in order to provide a solution for exchanging and re-using ontologies. To achieve this, Oyster implements the proposal for a metadata standard, so called Ontology Metadata Vocabulary (OMV) as the way to describe ontologies.

Oyster offers a user driven approach where each peer has its own local registry of ontology metadata and also has access to the information of others registries, thus creating a virtual decentralized ontology metadata registry. The Oyster client on its own (e.g. disconnected from the P2P network) will already provide added value to its users as it will give developers an overview and search facilities of his/her own ontology metadata stored in its local registry. The goal is a decentralized knowledge sharing environment using Semantic Web technologies that allows developers to easily share ontologies.

We argue that a decentralized system is the technique of choice, since it allows the maximum of individuality while it still ensures exchange with other users. Furthermore, ontologies are geographically distributed among the community, and developers are willing to share the information about the ontologies they created provided they do not have to invest much work in doing so, while at the same time they are able to maintain the ownership of their ontologies. A centralized approach allows reflecting long-term community processes in which some ontologies become well accepted for a domain or community and others become less important. However, both approaches could be combined to cover a variety of use cases.

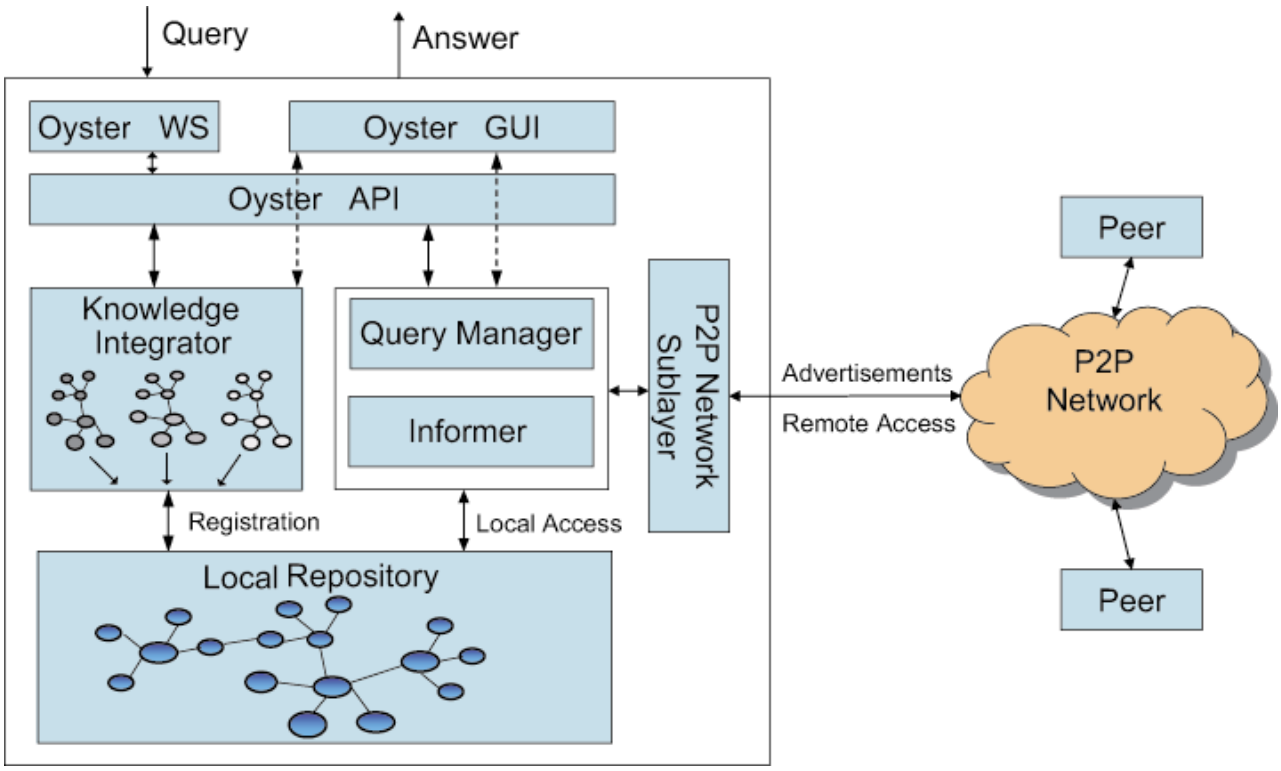

Figure 4.1: Overview of Oyster Architecture

4.1.1 Oyster Design

The Oyster system (Oyster system is freely available under `http://ontoware.org/projects/oyster2/`) was designed using a service-oriented approach, and it provides a well defined API. Accessing the registry functionalities can be done using directly the API within any application, invoking the web service provided or using the included java-based GUI as a client for the distributed registry. As part of the design, Oyster identifies an ontology metadata entry by the URI of the ontology it describes, therefore two ontology metadata entries are considered the same when the URI of both ontologies are the same. However, due to the distributed nature and potentially large size of the Peer-to-Peer network, two ontology metadata entries might refer to the same ontology but have different URI, in which case they are considered duplicates. The way Oyster handles such duplicates is described in the next section.

In Oyster, ontologies are used extensively in order to provide its main functions (register metadata, formulating queries, routing queries and processing answers).

Oyster Architecture

The high-level design of the architecture of a single Oyster node in the Peer-to-Peer system is shown in Figure 4.1.

In the following, we discuss the individual components of the system architecture.

The *Local Repository* of a node contains the metadata about ontologies that it provides to the network. It supports query formulation and processing and provides the information for peer selection. In Oyster, the Local Repository is based on KAON2 and it supports SPARQL as its query language. This component and the same component in KAONp2p are parts of the Ontology Repository Service in Figure 2.2.

The *Knowledge Integrator* component is responsible for the extraction and integration of knowledge sources (i.e. ontologies) into the Local Repository. Oyster supports automatic extraction of metadata for OWL,

DAML+OIL, and RDF-S ontology languages. This component is also in charge of how duplicate query results are detected and merged (see section 4.1.2).

The *Query Manager* is the component responsible for the coordination of the process of distributing queries. It receives queries from the user interface, API or from other peers. Either way it tries to answer the query or distribute it further according to the content of the query. The decision to which peers a query should be sent is based on the scope of the query (i.e. a specific set of peers or entire network) and optionally on the knowledge about the expertise of other peers. This query manager is also shared with KAONp2p introduced next.

The *Informer* component is in charge of proactively advertise the available knowledge of a Peer in the Peer-to-Peer network and to discover peers along with their expertise. This is realized by sending advertisements about the expertise of a peer. In Oyster, these expertise descriptions contain a set of topics (i.e. ontology domains) that the peer is an expert in. Peers may accept these advertisements, thus creating a semantic link to the other peer. These semantic links form a semantic topology, which is the basis for intelligent query routing (see section 3.2.3).

The *Peer-to-Peer network sub-layer* is the component responsible for the network communication between peers. It provides communication services for the data exchange with remote nodes, i.e. to propagate advertisement messages and to realize the access to remote repositories. In Oyster, we rely on an RMI-based implementation, however, other communication protocols would be possible as well. This component is also a component in KAONp2p distributed reasoner.

The API, WS and GUI components are discussed in the next section.

Oyster in NeOn architecture NeOn architecture consists of three main conceptual architectural layers:

1. **Basic infrastructure services** This layer contains all the services, which are required in for most ontology engineering tasks.
2. **Engineering components** This layer contains the main functionality of the NeOn engineering tool-kit like collaborative work support, population of ontologies from text sources.
3. **GUI components** that contain the user interfaces mainly for the engineering components e.g. editors, browsers, ...).

The basic infrastructure layer consists of a set of core services (e.g. Repository services, Registry services, etc). Oyster is an implementation of the registry services. As it is required, it is based on the OMV ontology meta model. Further it provides an API and a Web Service interface to query, create, and manipulate ontology metadata information according to the OMV model.

Additionally, as part of the future work (see section 4.1.4), Oyster will provide some engineering components that will rely on the registry services (e.g. Versioning).

Finally, Oyster also provides a GUI component that implements a user interface to the registry services.

4.1.2 Overview Of Oyster

Oyster was originally developed within the EU IST thematic network of excellence Knowledge Web `http://knowledgeweb.semanticweb.org/`. The original version `http://oyster.ontoware.org/` was implemented as an instance of the Swapster system architecture `http://swap.semanticweb.org/`.

For the current version, Oyster was re-engineered within the NeOn project as described below:

- The local node repository of Oyster is based on KAON2 instead of Sesame to fit with the general requirement for application and case study support by NeOn architecture.
- The query language supported is SPARQL instead of SeRQL, as SPARQL is the major query language in NeOn.

- The communication between peers is based on RMI instead of JXTA, because RMI are better suitable for cross-platform deployment and usage.

- In addition to provide a user interface (GUI component), the current version of Oyster provides a set of APIs and a web service for exposing the registry services.

In the rest of this section we summarize the functionalities provided by Oyster and the alternatives it provides to interact with the registry.

Oyster API overview

In this section we provide an overview of the Oyster API.

1. **Register Metadata** Ontology Metadata can be registered through Oyster API either by specifying the property-value pairs of the ontology metadata or by importing ontology files in order to automatically extract the ontology metadata available. For the automatic extraction, Oyster supports the OWL , DAML+OIL , and RDF-S ontology languages. The ontology metadata entries are aligned and formally represented according to two ontologies: (1) the proposal for a metadata standard OMV that describes the properties of the ontology, and (2) a topic hierarchy (i.e. DMOZ) that describes specific categories of subjects to define the domain of the ontology.

2. **Update Metadata** Oyster API provides the means to update the ontology metadata stored locally. The ontology metadata will be modified by specifying the new set of property-value pairs or by importing again the ontology file with updated metadata information. Note that the ontology file is not a new version of the ontology, i.e. the semantics of the ontology is the same, only the metadata has changed (e.g. comments, title, etc.) or added (e.g. version information, authors, etc.).

3. **Search Metadata** The registry can be searched using the API by specifying the search criteria in terms of the two ontologies (e.g. OMV and topic hierarchy). This means that the search can refer to any property of OMV like URI, name, acronym, resourceLocator, etc. or they may refer to topic terms. Additionally, the registry can be searched by time conditions (i.e. ontology that has changed since a particular date).

   Optionally, when searching the registry, it is possible to specify if the search is over a single specific peer (e.g. their own computer, or a certain peer because this peer is known as a big provider of information), or over a specific set of peers (e.g. all the members of a specific organization), or over the entire network of peers (e.g. when the user has no idea where to search). In the latter case, queries are routed automatically through the network depending on the expertise of the peers, describing which topic of the topic hierarchy a peer is knowledgeable about. In order to achieve this expertise based routing, a matching function determines how closely the semantic content of a query matches the expertise of a peer (see section 3.2.3).

   The search returns a list of one or many metadata entries, and if the results includes many entries of the same metadata (i.e. metadata entries in different peers with the same URI), then a merged representation that combines the knowledge from the individual and potentially incomplete items is returned in the result.

4. **Get Peers** The Oyster API can be used to retrieve the set of all active Peers in the network. This set of Peers then can be evaluated to restrict the search to a certain set of Peers as described above.

5. **Get Peer Expertise** This method of the API allows to retrieve the complete list of all ontologies registered in a specific Peer.

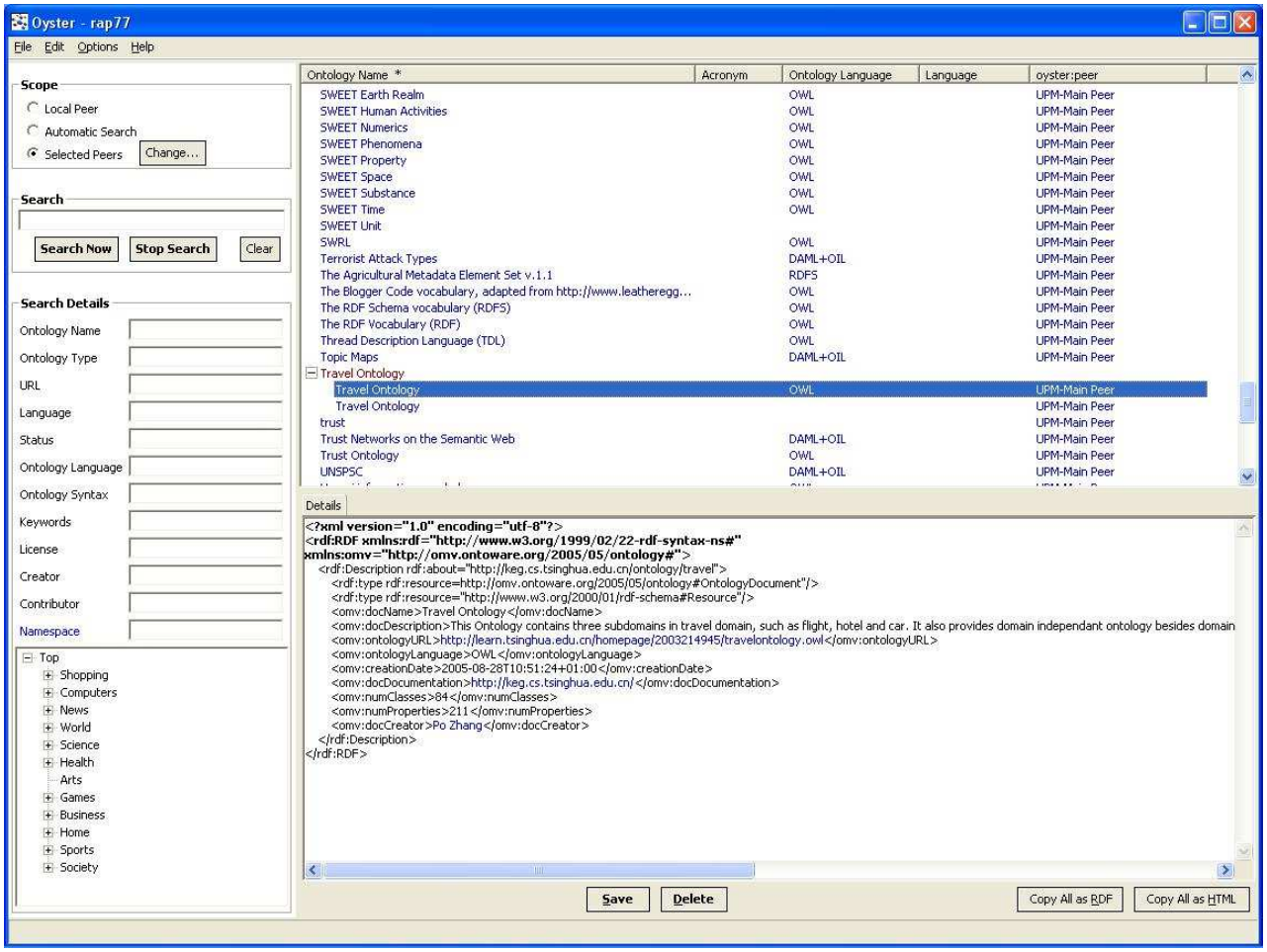

Figure 4.2: Oyster GUI

Oyster GUI overview

Oyster GUI provides a ready to use client interface to the distributed registry. It implements most of the functionalities provided by the API and offers to the final user a way to interact with the registry.

1. **Publishing and Importing Metadata** Oyster GUI enables users to register metadata about ontologies manually and also to import ontology files to extract the ontology metadata available (as described in section 4.1.2) and let the user specify the missing values.

2. **Formulating Queries** As shown in the left pane of the figure 4.2, users can use the GUI to search the registry for ontology metadata by means of simple keyword searches, or more advanced semantic queries like SPARQL queries. Queries are formulated in terms of these two ontologies as described in section 4.1.2. The GUI includes by default some of the most common properties used for searching ontologies like name, acronym, ontology language, etc. Additionally, it also includes a way to search ontologies by topic terms.

3. **Routing Queries** As shown in the upper left pane of the figure 4.2, the Oyster GUI provides the means for the users to specify if they want to query a single specific peer, a specific set of peers, or the entire network of peers (see section 4.1.2).

4. **Processing results** The results matching a query are presented in a result list (c.f. upper right pane in figure 4.2). The answer of a query might be very large and may contain many duplicates due to the

distributed nature and potentially large size of the Peer-to-Peer network. Such duplicates might not be exact copies because of the semi structured nature of the metadata, so the ontologies are used again to measure the semantic similarity between different answers and to detect apparent duplicates. For Oyster, duplicates are ontology metadata entries which refer to the same ontology, but are modelled as different resources (i.e. using different URIs). In order in recognize two entities as being duplicates, Oyster applies specific similarity functions.

In detail, we compiled a set of specific ontology facts used to assess the similarity between two instances (e.g. URL, name, version, type, domain, creationDate). For each of these facts we use different individual similarity functions depending on the type of information it represents. Then, from the results of each individual similarity function, we obtain an overall value with an aggregated similarity function, using a weighted average over the individual functions. These weights were assigned based on experiments with sample data. Using the overall value, we consider duplicates those pairs of resources whose similarity is larger than a certain threshold.

Then, the result list provides all the possible duplicates as a group for the user to visualize them. The details of particular results are shown in the lower right of the figure 4.2. Additionally users can save the results of a query into their local repository for future use.

Oyster WS overview

Oyster WS encapsulates the Oyster API as a Web Service so it can be used within any service oriented application. The functionality provided by the Web Service is basically the same as the functionality provided by the API described above.

4.1.3 Experience

In this section we present the evaluation of the first version of Oyster regarding usage statistics in order to show the usability of the system. For that experiment, the evaluated version of Oyster that can be downloaded from `http://oyster.ontoware.org` requests the user permission when it is installed for sending usage statistics (i.e. logs) to a central server. This information has been collected during a period time of 6 months, in randomly chosen days of the months. We logged user behavior and actions in a time window of 15 minutes. During this time period the peers issued 333 queries overall and received 29076 distinct results in terms of RDF statements, which corresponds to about 3000 ontology metadata entries.

A total of 250 ontology metadata entries were shared, among 42 peers, with an average of 6 ontologies per peer. However, the distribution had a high variance: While only three peers shared 85% of the total content, a lot of peers provided only one or two entries or were ẗree-riding..

The analysis of the type of query the users sent revealed that around 50% of the queries were using at least one of the two ontologies used in Oyster. OMV properties (e.g. name, type) were used in more that the 27% of the queries and in over 23% of the queries the users asked for topics of the DMOZ topic hierarchy. Based on that analysis we could determine that the preferred properties for searching ontologies by the users are the domain, the name and the ontology language. The other 50% of the queries were general searches (i.e. get all the available ontologies).

Due to the limited size of the network and the lack of domain information on most of the ontology metadata entries, the evaluation of the expertise based peer selection couldn't be shown. However, in simulation experiments with larger peer networks with thousands of peers, it has been shown improvements in order of one magnitude in terms of recall of resources and relevant peers[HSvH04]. For evaluation of other Swapster based system we refer the reader to [HBE+04].

4.1.4 Conclusions and Future Work

As we summarized in this section, Oyster implements a distributed ontology metadata registry that exploits semantic web technologies in order to provide a solution for sharing and re-using ontologies. Oyster tackles the problems of the heterogeneity, distribution and diverse ownership of the ontologies as well as the lack of sufficient metadata by implementing a proposed standard for metadata for describing ontologies.

Oyster allows a flexible interaction with the metadata registry by providing a set of APIs that can be used within any java application, a Web Service component that can be used within any web application and a ready to use GUI that can be used directly by the users to interact with the registry.

Finally, Oyster future work addresses many challenges. In particular, those related to NeOn includes versioning management support, handle modularization information, manage and support other network relations between ontologies (i.e. Extension, specialization, etc.), manage ontology evaluation information, support and exploit review and rating information.

Additionally, our future work includes coordinate the integration of Oyster with *Watson* infrastructure developed by the Open University. Watson provides a centralized scalable mechanism for discovering, selecting, and accessing ontologies that are normally distributed over the Web. The proposal for the integration of Oyster with Watson is as follows: Since Oyster is a distributed ontology metadata registry using a peer-to-peer architecture to store and manage its content, the outcomes of Watson's validating and analytic tools may act as a dedicated 'Oyster peer'. This 'Oyster peer' would then expose some (agreed upon) information about crawled and discovered ontologies that Watson currently maintains in its "Ontology and Metadata DB".

The interaction between both systems will be in both directions. On the one hand, Watson will provide a service in charge of wrap Watson information about discovered ontologies using Oyster's registry format (i.e. OMV) in addition to submit the outcomes of Watson crawling, analysis, and/or validation to Oyster registry. On the other hand, at regular intervals, Watson will get new metadata from Oyster network.

Many of the challenges for the future work mentioned earlier will be coordinated with Watson. For example, the management of ontology versions will require a good coordination between the versions of ontologies stored in Watson and the ontology metadata stored in Oyster. Additionally, information about modularization or other networked relations will need to be synchronized.

4.2 KOANp2p – A Decentralized Ontology Query Answering System

In this section we propose an infrastructure, in which we address these challenges in an integrated manner to realize query answering over heterogeneous OWL DL ontologies distributed over multiple nodes. This infrastructure builds on, extends and integrates a number of prior techniques we have developed in the context of distributed metadata management and reasoning. In order to deal with the coordination the nodes we rely on metadata about nodes and their provided resources to support their interoperation and coordination. In our approach, nodes advertise descriptions of their resources and can thus establish acquaintances with other nodes. Each node maintains the metadata about available resources in its own local registry in a completely decentralized manner. Acquainted nodes can then share data and coordinate their interaction. To be able to integrate heterogeneous ontologies, we propose an expressive mapping formalism in which mappings are expressed as correspondences between queries over source and target ontologies. An important aspect of this mapping formalism is that it does not rely on the notion of a global ontology, as known in classical integration systems based on LAV or GAV (Local-As-View, Global-As-View). Instead, mappings can be formulated in any direction between arbitrary nodes. We have implemented the reasoning infrastructure in KAONp2p, extending the OWL reasoner KAON2.

4.2.1 Overview of KAONp2p

In this section, we present a brief overview of KAONp2p (The system is freely available for download at `http://kaonp2p.ontoware.org/`.). The general architecture of a single KAONp2p node interacting with

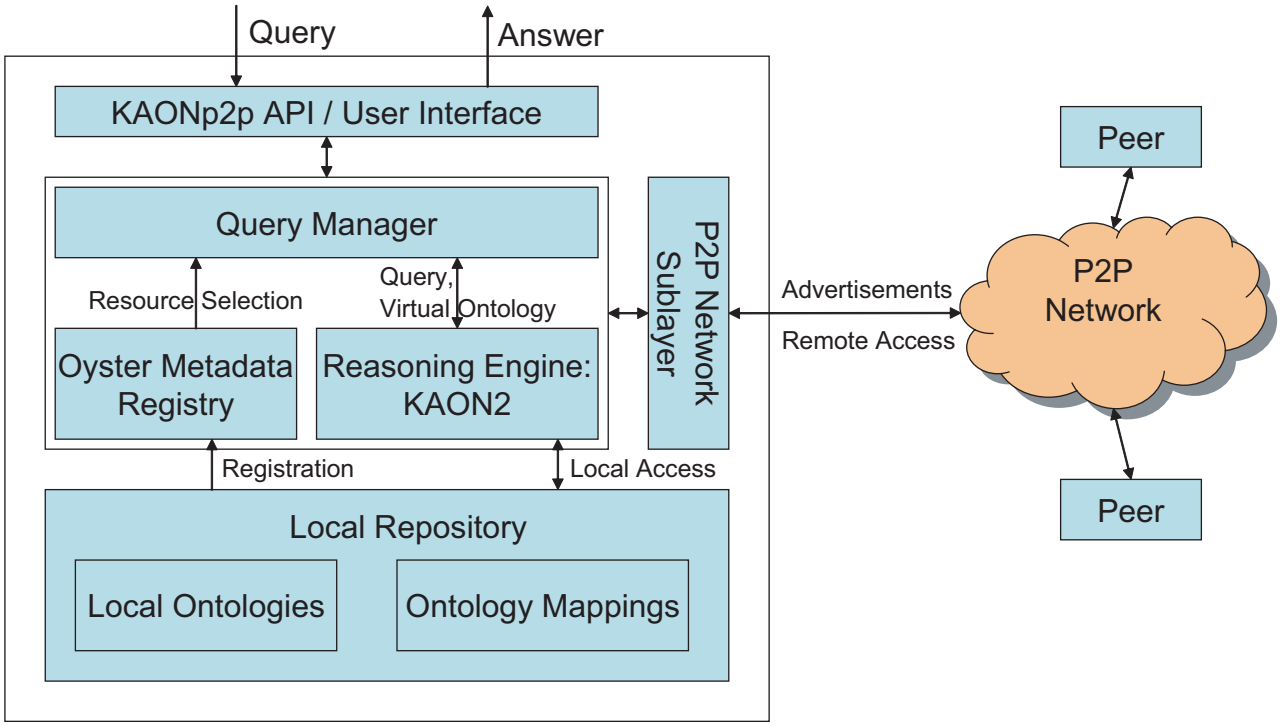

Figure 4.3: Overview of KAONp2p Architecture

remote nodes is shown in Figure 4.3. In the following, we discuss the individual components of the system architecture.

The *Local Repository* of a node contains the ontologies it provides to the network along with mappings that relate heterogeneous ontologies available in the network. It is important to note that mappings are first-class objects in the system that can be shared with other nodes.

The *Oyster Metadata Registry* One important concept to deal with the coordination of a distributed information system is the concept of metadata. Metadata is explicitly managed data about the system elements to support their interoperation and coordination. KAONp2p uses Oyster system [PH05a] as metadata registry. We refer readers to Section 3.2 for detailed description of this metadata registry.

The *Query Manager* is the component responsible for answering queries against the available ontologies in the network. As queries we consider conjunctive queries over OWL DL ontologies. The query process can be divided into two steps:

1. **Resource selection.** The goal of the resource selection is to identify resources in the network that are relevant to answer a particular user query. This process is governed by selection algorithm that matches the subject of the query against resource descriptions stored in the *Oyster Metadata Registry*. The Oyster Metadata Registry maintains metadata about resources available (i.e. peers, ontologies, and mappings), which may be accessible either locally or remotely in the network. The resources are described using the metadata ontology described in Section 3.2. The result of the selection process is a “virtual ontology” that logically integrates relevant ontologies and mappings required to mediate between the heterogeneous ontologies in the network, represented using the mapping formalism described in Section 4.2.2. We offer three options for resource selection: (1) a manual selection, where the user can choose the resources relevant to his query, (2) a trivial selection that includes all resources known in the registry, and (3) automated selection that is introduced in detail in Section 3.2.3. In addition to the selected ontologies, available mappings are identified that relate the heterogeneous remote ontologies to the target ontology against which the query is expressed. The relevant resources are integrated in the virtual ontology, which will be used in the second step of query answering described

in the following Section.

2. **Query answering.** In the second step, the query is evaluated against the virtual ontology within the *Reasoning Engine*. In our implementation, we rely on KAON2 as a reasoner. The reasoning algorithms of KAON2 do not require the integrated ontologies to be materialized locally, instead the distributed ontologies still reside on the remote server, and only relevant parts need to be retrieved to the local node. The details of this process will be explained in Section 4.2.2.

The *Peer-to-Peer network sub-layer* provides communication services for the data exchange with remote nodes, i.e. to propagate advertisement messages and to realize the access to remote ontologies. In the implementation of KAONp2p, we rely on an RMI-based implementation, however, other communication protocols would be possible as well.

By means of the *KAONp2p user interface and API*, users can pose queries, receive answers to queries and control the configuration of the system. While in the user interface queries queries are formulated in a visual manner, they are passed to the API as conjunctive queries in SPARQL.

4.2.2 Mappings in KAONp2p

To enable interoperability between nodes in large distributed information systems based on heterogeneous ontologies, it is necessary to specify how the data residing at a particular node corresponds to data residing at another node. This is formally done using the notion of a mapping. There are three lines of work connected to the problem of mapping: (1) identifying correspondences between heterogeneous data sources, (2) representing these correspondences in an appropriate mapping formalism, and (3) using the mappings for a given integration task. We here focus on the latter two important problems once the correspondences between data sources are known: those of representing the mappings using an appropriate formalism and using them for the task of query answering over heterogeneous data sources.

We follow the general framework of [Len02] to formalize the notion of a mapping system for OWL DL ontologies, where mappings are expressed as correspondences between conjunctive queries (We denote a conjunctive query as $q(\mathbf{x}, \mathbf{y})$, with $\mathbf{x}$ and $\mathbf{y}$ sets of *distinguished* and *non-distinguished* variables, respectively.) over ontologies. The components of this mapping system are the source ontology, the target ontology, and the mapping between them.

Definition 3 (Ontology Mapping System) *An ontology mapping system $\mathcal{MS}$ is a triple $(\mathcal{S}, \mathcal{T}, \mathcal{M})$, where*

- *$\mathcal{S}$ is the* source ontology ontology*,*
- *$\mathcal{T}$ is the* target ontology ontology*,*
- *$\mathcal{M}$ is the mapping between $\mathcal{S}$ and $\mathcal{T}$, i.e. a set of assertions $q_S \rightsquigarrow q_T$, where q_S and q_T are conjunctive queries over $\mathcal{S}$ and $\mathcal{T}$, respectively, with the same set of distinguished variables $\mathbf{x}$, and $\rightsquigarrow \in \{\sqsubseteq, \sqsupseteq, \equiv\}$.*

An assertion $q_S \sqsubseteq q_T$ is called a sound *mapping, requiring that q_S is contained by q_T w.r.t. $S \cup \mathcal{T}$; an assertion $q_S \sqsupseteq q_T$ is called a* complete *mapping, requiring that q_T is contained by q_S w.r.t. $\mathcal{S} \cup \mathcal{T}$; and an assertion $q_S \equiv q_T$ is called an* exact *mapping, requiring it to be sound and complete.*

Let us discuss the expressiveness in terms of the ontology language, the query language and the assertions. The expressiveness of conjunctive queries corresponds to that of the well-known select-project-join queries in relational databases. Two typical approaches to specify mappings are the *global-as-view* (GAV) approach, where elements of the target are described in terms of queries over source, and the *local-as-view* (LAV) approach, where elements of the source are described in terms of queries over target. Our mapping

system subsumes the approaches of GAV, LAV. In fact, it corresponds to the GLAV approach, which is more expressive than GAV and LAV combined.

In [HM05] the semantics of the mapping system has been defined by translation into first-order logic. We here only discuss the intuitions behind the semantics of the main inference task for $\mathcal{MS}$, i.e. computing answers for a conjunctive query $Q(\mathbf{x}, \mathbf{y})$ w.r.t. $\mathcal{MS}$.

As query answering within such a mapping system is undecidable in this generality, we have identified classes of mappings that introduce restrictions required to attain decidability. These restricted, but still very expressive mappings, can be expressed either directly in OWL DL, or in OWL DL extended with the so-called *DL-safe* subset of the Semantic Web Rule Language (SWRL) [MSS04].

The first class of mappings captures the mappings that can be directly expressed in OWL DL. This is the case if q_s and q_t are of the form $P_s(\mathbf{x})$ and $P_t(\mathbf{x})$, where P_s and P_t are DL predicates: If q_s and q_t are of the form $P_s(x)$ and $P_t(x)$ and P_s, P_t are DL concepts, the mapping corresponds to the equivalent concept inclusion axiom. If q_s and q_t are of the form $P_s(x_1, x_2)$ and $P_t(x_1, x_2)$, with P_s and P_t are abstract or concrete roles, the mapping corresponds to the equivalent role inclusion axiom.

The second class of mappings captures the so-called *DL-safe Mappings.* Let us consider a sound mapping $q_S \sqsubseteq q_T$ (For a complete mapping $q_S \sqsupseteq q_T$, the situation is analogous, with the roles of q_S and q_T reversed.) with the assertion $\forall \mathbf{x} : q_T(\mathbf{x}, \mathbf{y}_T) \leftarrow q_S(\mathbf{x}, \mathbf{y}_S)$. In our restriction, we disallow the use of non-distinguished variables in the query q_T, i.e. restrict the assertions to the form $\forall \mathbf{x} : q_T(\mathbf{x}) \leftarrow q_S(\mathbf{x}, \mathbf{y}_S)$ (Please note that these assertions correspond to SWRL rules) and require the query q_S to be DL-safe, thus limiting the applicability of the rules to known individuals. Thus obtained mappings correspond to (one or more) DL-safe rules, for which efficient algorithms for query answering are known [MSS04].

4.2.3 Query Answering in KAONp2p

We now show how to use an ontology mapping system for query answering in an *ontology integration system*, whose main task is to provide integrated access to a set of distributed source ontologies. The integration is realized via a mediated target ontology through which we can query the local ontologies.

Definition 4 *For a set of local source ontologies $\mathcal{S}_1, \ldots, \mathcal{S}_n$, a target ontology $\mathcal{T}$ and corresponding mapping systems $\mathcal{MS}_1, \ldots, \mathcal{MS}_n$ with $\mathcal{MS}_i = (\mathcal{S}_i, \mathcal{T}, \mathcal{M}_i)$, an* ontology integration system *$\mathcal{IS}$ is again a mapping system $(\mathcal{S}, \mathcal{T}, \mathcal{M})$ with $\mathcal{S} = \bigcup_{i \in \{1 \ldots n\}} \mathcal{S}_i$ and $\mathcal{M} = \bigcup_{i \in \{1 \ldots n\}} \mathcal{M}_i$. The main inference task for $\mathcal{IS}$ is to compute answers of $Q(\mathbf{x}, \mathbf{y})$ w.r.t. $\mathcal{S} \cup \mathcal{T} \cup \mathcal{M}$, for $Q(\mathbf{x}, \mathbf{y})$ a conjunctive query over $\mathcal{T}$.*

Please note that because of the absence of a global ontology, this form of ontology integration system can be directly applied to decentralized integration: For our set of autonomous nodes, each relying on some local ontology, and a set of mappings that relate the local ontology to those of other nodes, an ontology integration system $\mathcal{IS} = (\mathcal{S}, \mathcal{T}, \mathcal{M})$ can easily be constructed for each individual node, where $\mathcal{S}$ consists of the ontologies of the remote node to be integrated, $\mathcal{T}$ is the ontology of the local node, and $\mathcal{M}$ consists of the individual mappings systems describing the correspondences between the local ontology with remote ontologies. This construction is performed during the resource selection process described in Section 3.2.3, which selects the relevant source ontologies $\mathcal{S}$ and the required mappings $\mathcal{M}$.

We now discuss how to compute answers to a conjunctive query $Q(\mathbf{x}, \mathbf{y})$ in an ontology integration system $\mathcal{IS}$. The algorithm is based on the correspondence between description logics and disjunctive datalog from [HMS04]. Given an OWL DL knowledge base KB (without nominals) extended with DL-safe rules, a positive disjunctive datalog program DD(KB) is produced, which entails exactly the same set of ground facts as KB, i.e. $KB \models A$ if and only if DD$(KB) \models A$, for A a ground fact. Thus, query answering in KB is reduced to query answering in DD(KB), which can be performed efficiently using the techniques of (disjunctive) deductive databases.

Query answering can be performed in time exponential in the size of KB. Furthermore, as shown in [HMS05], the data complexity of these algorithms (i.e. the complexity assuming the size of the schema is fixed) is NP-complete, or even P-complete if disjunctions are not used.

Based on these results, we are able to perform query answering in the ontology integration system by converting it into a disjunctive datalog program. The source ontology, target ontology and the mappings are converted into a disjunctive datalog program, and the original query is answered in the obtained program $\mathrm{DD}(\mathcal{S} \cup \mathcal{T} \cup \mathcal{M})$. By the results from [HMS04, MSS04], it is easy to see that the algorithm exactly computes the answer of $Q(\mathbf{x}, \mathbf{y})$ in $\mathcal{IS}$.

From the above definition, one might get the impression that our algorithm requires that all source and target ontologies must be physically integrated into one mapping system in order to answer queries. This is, of course, not the case. More concretely, to compute $\mathrm{DD}(\mathcal{S} \cup \mathcal{T} \cup \mathcal{M})$, it is necessary to physically integrate the TBox part of $\mathcal{S}$, $\mathcal{T}$ and $\mathcal{M}$. Since the TBox are typically much smaller than the data, this does not pose practical problems. Accessing actual data sources (i.e. the ABoxes) is then governed by the chosen strategy for evaluating the datalog program. In practice, we only need to access the extensions of those predicates from remote nodes that are actually relevant for the datalog program.

4.2.4 Evaluation

In this section we present experimental results for the evaluation of the KAONp2p infrastructure. In this evaluation we focus on the second part of the query processing, i.e. the actual query answering against a set of relevant resources. Please note that for this second part, the number of nodes will be typically much smaller than the number of nodes under consideration for the resource selection. For evaluation regarding the first step of resource selection (with several thousands of nodes), we refer to evaluations reported in [HBE+04] and [HSvH04].

In our first experiment we have used the Lehigh University Benchmark (LUBM) [GPH05] to evaluate the performance and scalability in terms of the distribution. We deployed eight physically distributed KAONp2p nodes, each of which holding a different automatically generated data set according to the LUBM ontology (`http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl`) describing instance data of one university, approximately 8MB of OWL/RDF data.

We used queries selected from the LUBM benchmark based on different complexities for the experiments and format them in SPARQL query language. The first query “Q1” was to seek all the graduate student; the “Q2” aimed to search the publications with their corresponding authors who were assistant professors; the last query “Q3” wanted to find out all the graduate students, universities and departments in such a relationship – the graduate students were members of the departments that were sub-organizations of the universities where the graduate students got their undergraduate degree.

```
Q1: SELECT ?x WHERE { ?x rdf:type lubm:GraduateStudent }

Q2: SELECT ?x ?y WHERE { ?x rdf:type lubm:AssistantProfessor .
    ?y rdf:type lubm:Publication . ?y lubm:publicationAuthor ?x}

Q3: SELECT ?x ?y ?z WHERE { ?x rdf:type lubm:GraduateStudent .
    ?y rdf:type lubm:University . ?z rdf:type lubm:Department .
    ?x lubm:memberOf ?z . ?z lubm:subOrganizationOf ?y .
    ?x lubm:undergraduateDegreeFrom ?y }
```

We then used an additional node to perform the queries against the knowledge bases of 1..8 manually selected nodes. Figure 4.4 shows the results of the execution times for the query answering. The main observation is that in this particular scenario, the time for answering queries increases approximately linearly with the number of nodes and thus the size of the data set. The additional degree of distribution does not incur a performance penalty.

In a second experiment we evaluated the additional costs introduced by the heterogeneity between the nodes. In this experiment, we deployed two nodes, one with an automatically generated LUBM data set, another one with an SWRC (`http://ontoware.org/projects/swrc/`) data set, containing real life data from the University of Karlsruhe (`http://www.aifb.uni-karlsruhe.de/viewAIFB_OWL.owl`). Further, we defined mappings according to our mapping formalism to relate the LUBM ontology with the SWRC ontology in both directions. We then used an additional node to perform queries against the node providing the

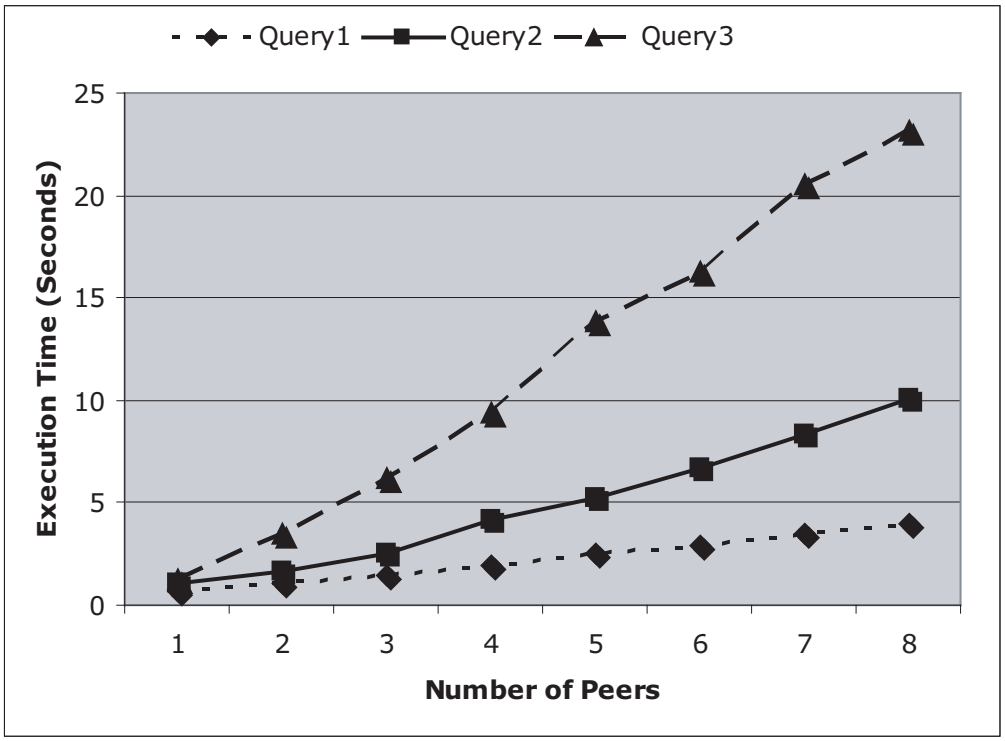

Figure 4.4: Evaluation Results for LUBM 1..8

SWRC data set as source ontology, where in a first case the query is expressed in terms of the same target ontology in a homogeneous setting (SWRC - SWRC) (For SWRC as target ontology we rephrased the three queries above in terms of SWRC.) and in a second case the query is expressed against a different target ontology (LUBM - SWRC) in a heterogeneous setting. We repeated the experiment queries against the node providing the LUBM data set as source ontology, which we again queried with LUBM as target ontology (LUBM - LUBM) and SWRC as target ontology (SWRC - LUBM). Figure 4.5 shows the results for the query execution times. We observe that the time needed for query answering increases only slightly for the case where the source and target ontologies differ and thus mappings are required. The reason lies in the fact that the mappings are only used in the computation of the datalog program, which is neglectable compared to the evaluation of the program. This makes our approach especially applicable for scenarios where mappings between heterogeneous ontologies are required.

Summarizing, the evaluation results show that in approach the performance of query answering is essentially dominated by the size of the data, and only slightly affected by the degree of distribution and heterogeneity. In fact, it shows a performance comparable to a setting where data resides on a single, homogeneous node.

4.2.5 Conclusions and Future Work

In this section we have introduced KAONp2p, a Peer-to-Peer system for query answering over distributed ontologies in decentralized networks. This infrastructure addresses (i) the coordination of multiple nodes using metadata about the provided resources managed in a decentralized registry, (ii) the mediation between heterogeneous ontologies via an expressive mapping formalism as well as corresponding reasoning algorithms for query answering.

The query processing follows a two-step process consisting of: (1) the selection of relevant resources based on metadata managed in a metadata registry, (2) query answering against relevant resources, which are integrated using a virtual ontology, which logically imports relevant ontologies and mappings. This virtual

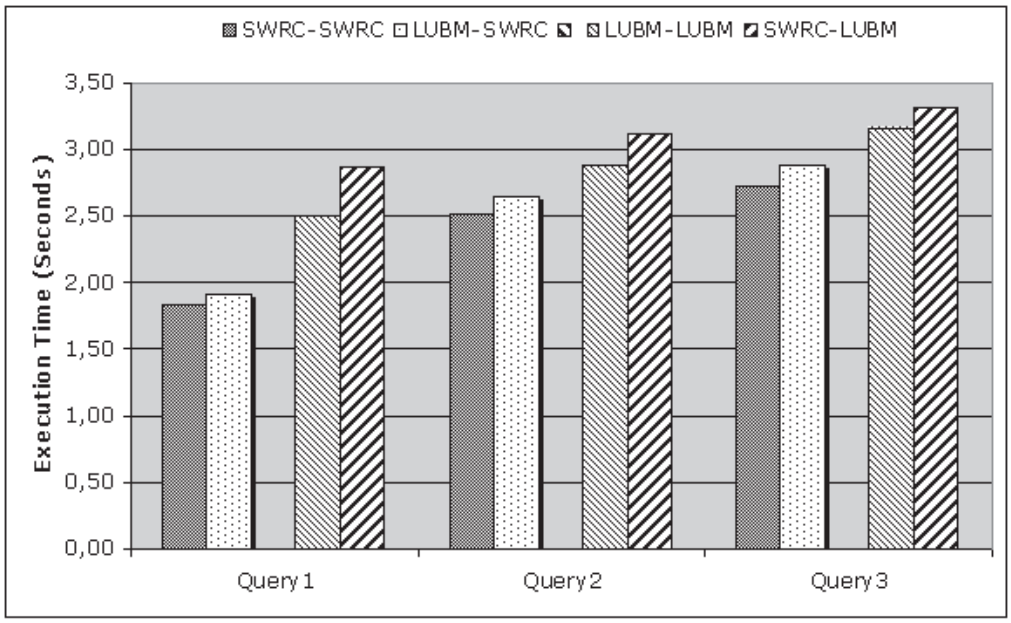

Figure 4.5: Cost of Mappings between Heterogeneous Ontologies

integration provides global model semantics as if all ontologies were integrated locally. Practically, the distributed ontologies still reside on the remote nodes, and only the parts relevant for answering the query need to be retrieved to the local node. Our evaluation results show that the approach is very promising as performance of query answering is essentially dominated by the size of the data, and only slightly affected by the degree of distribution and heterogeneity. In fact, the performance is comparable to settings where the data resides on a single, homogeneous node.

4.3 KAONWeb – An Infrastructure for Optimizing Distributed Query Answering over Networked ontologies

In this section, we present a query answering infrastructure for networked ontologies with a novel adaptable and hybrid algorithm that is particularly optimized for a distributed scenario where multiple ontologies are used and mappings are present. An actual implementation of the algorithms in the *KAONWeb* system is evaluated and proves to be favorable to prior algorithms not taking the optimization. We also describe important aspects of the KAONWeb system (The experimental KAONWeb system can be downloaded at: `http://www.aifb.uni-karlsruhe.de/WBS/ywa/kaonweb/KAONWeb.zip`) which provides a distributed system for ontology query answering. Different from KAONp2p, KAONWeb is developed to support the distributed scenario where the data are distributed with central control and coordination, therefore the major task for KAONWeb is to provide very efficient distributed query answering.

4.3.1 Query Answering in KAONWeb

Queries such as Example 1 are answered using distributed query answering algorithms. The algorithms are used in the context of a distributed ontology integration system, which provides the structure used in the algorithms, such as mappings between ontologies.

Distributed Ontology Integration System

Mappings between ontologies are core elements of ontology integration systems. We follow the general framework of [Len02] to define mappings for $\mathcal{SHIN}(\mathbf{D})$ ontologies, where mappings are expressed as correspondences between conjunctive queries[1] over ontologies. This mapping system is composed of a source ontology, a target ontology, and a mapping between them.

Given the mapping system in Definition 3, in [HM05] the semantics of the mapping system has been defined by translation into first-order logic. We here only discuss the intuitions behind the semantics of the main inference task for $\mathcal{MS}$, i.e., computing answers for a conjunctive query $Q(\mathbf{x}, \mathbf{y})$ w.r.t. $\mathcal{MS}$. To understand the intuition of computing answers, we briefly recall the semantics of query answering as defined in [Len02]: An answer of a conjunctive query $Q(\mathbf{x}, \mathbf{y})$ w.r.t. a knowledge base KB is an assignment θ of individuals to distinguished variables, such that $KB \models Q(\mathbf{x}\theta, \mathbf{y})$. Thus, the intuitive reading of this semantics is that an answer of a query needs to be entailed by the source ontology $\mathcal{S}$, the target ontology $\mathcal{T}$ and the mappings $\mathcal{M}$. This semantics is equivalent to the usual model theoretic semantics (e.g., in [CDL01]) based on local and global models, where a query answer must be an answer in *every* global model. The decidability of query answering within such a mapping system is discussed in [HM05].

We have realized that it is possible to refine the performance of query answering over networked ontologies in distributed environment by distributing the query tasks to the distributed nodes. We herein define the notion of *distributed ontology integration system* in such a distributed scenario:

Definition 5 (Distributed Ontology Integration System) *Given a set of local source ontologies $\mathcal{S}_1, \ldots, \mathcal{S}_n$, a target ontology $\mathcal{T}$ and corresponding mapping systems $\mathcal{MS}_1, \ldots, \mathcal{MS}_n$ with $\mathcal{MS}_i = (\mathcal{S}_i, \mathcal{T}, \mathcal{M}_i)$, a* distributed ontology integration system *$\mathcal{IS}$ is a mapping system $(\mathbf{S}, \mathcal{T}, \mathbf{M})$ where $\mathbf{S} = \{\mathcal{S}_1, \ldots, \mathcal{S}_n\}$ is a set of source ontologies and $\mathbf{M} = \{\mathcal{M}_1, \ldots, \mathcal{M}_n\}$ is a set of mappings that relate $\mathbf{S}$ to $\mathcal{T}$, and $\mathcal{T} \notin \mathcal{S}$. The main inference task for $\mathcal{IS}$ is to compute answers of Q w.r.t. $\mathcal{S} \cup \mathcal{T} \cup \mathcal{M}$, for Q a conjunctive query over $\mathcal{T}$. In $\mathcal{IS}$, given a set of distributed nodes N where the elements of $\mathcal{IS}$ reside, a location function has the signature* $\mathsf{Loc} : (\mathbf{S} \cup \mathcal{T} \cup \mathbf{M}) \rightarrow N$ *and establishes* $N = \mathsf{Loc}(\mathbf{S})$.

Because of the absence of a global ontology, this form of ontology integration system can be directly applied to decentralized integration: Assume we have a set of autonomous nodes, each relying on a local ontology, and a set of mapping systems that relate the local ontology to those of other nodes. An ontology integration system $\mathcal{IS} = (\mathbf{S}, \mathcal{T}, \mathbf{M})$ can easily be constructed for each individual node, where $\mathbf{S}$ consists of the ontologies of the remote node to be integrated, $\mathcal{T}$ is the ontology of local node, and $\mathbf{M}$ consists of the individual mappings describing the correspondences between the local ontology with remote ontologies.

Nevertheless, integrating and query *all* the ontologies to cover *all* the dependencies between ontologies is not going to scale over a large set of distributed nodes, we need to develop an appropriate approach to distribute the query answering tasks and optimize the performance for the distributed scenario. The distributed query answering can be achieved by splitting the $\mathcal{IS}$ into several fragments and perform query answering over these fragments, which we denote *components* of $\mathcal{IS}$ here for the fragments. We first present an example setup to explain how these components can be queried in a distributed manner.

Example Setup

The Figure 4.6 depicts a typical scenario query answering in $\mathcal{IS}$, which consists of three distributed nodes with source ontologies $\mathcal{S}_1$, $\mathcal{S}_2$, $\mathcal{S}_3$ and $\mathcal{S}_4$, and the mapping $\mathcal{M}_1$, $\mathcal{M}_2$ and $\mathcal{M}_3$ located on the second distributed node. Please note that any ontology in the networked scenario can be target, source ontology or mapping. We denote $\mathcal{T}$ as target ontology and $\mathcal{M}$ as mapping in Figure 4.6 only for straightforward presentation.

If we query $\mathcal{T}$ on node 1, $\mathcal{S}_2$ and $\mathcal{S}_3$ are source ontologies that are defined as coupled with each other (presented as dashed line) and mapped to $\mathcal{T}$ via $\mathcal{M}_1$ and $\mathcal{M}_2$ (presented as solid line), respectively. This

[1] We denote a conjunctive query as $q(\mathbf{x}, \mathbf{y})$, with $\mathbf{x}$ and $\mathbf{y}$ sets of *distinguished* and *non-distinguished* variables, respectively.

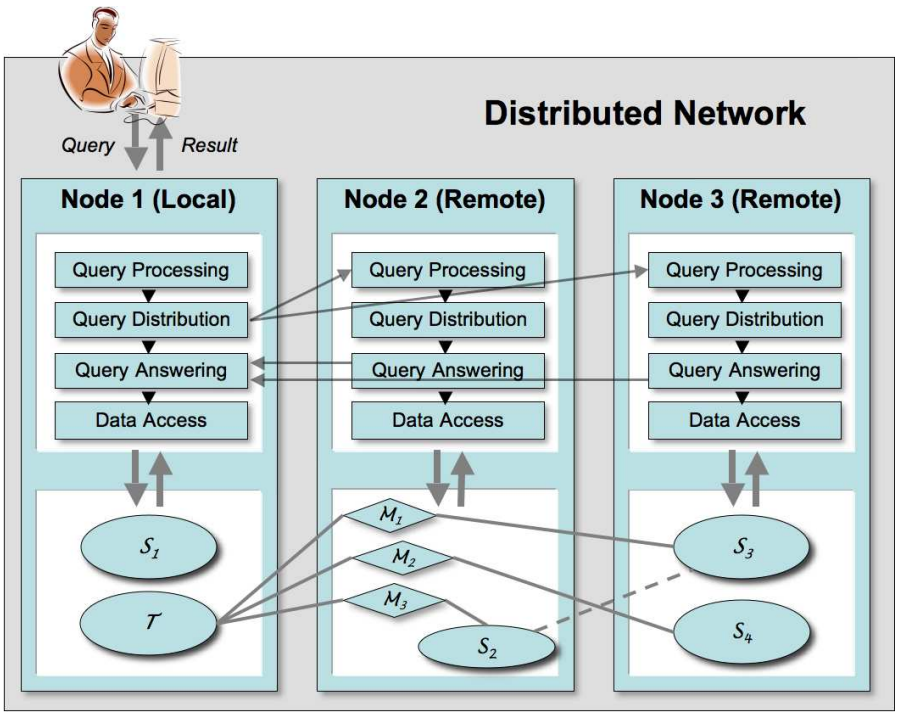

Figure 4.6: Scenario of hybrid query answering. The user sends the query to the distributed query answering system on node 1 that processes the query, splitting the $\mathcal{IS}$, queries over corresponding ontologies, collects the results and sends them back to the user.

is a very typical component of $\mathcal{IS}$ and can be established as: $(\{\mathcal{S}_2, \mathcal{S}_3\}, \mathcal{T}, \{\mathcal{M}_3, \mathcal{M}_1\})$. $\mathcal{S}_4$ is defined as a self-contained ontology and mapped to $\mathcal{T}$ via $\mathcal{M}_3$. This case contains a single-element of $\mathcal{IS}$: $(\mathcal{S}_4, \mathcal{T}, \mathcal{M}_2)$. A special case also occurs when $\mathcal{S}_1$ is also defined as a self-contained ontology but $\mathcal{S}_1$ and $\mathcal{T}$ share a same schema. This kind of component can be presented as: $(\{S_1\}, \mathcal{T}, \emptyset)$. If we execute a conjunctive query, the system automatically adapts to the corresponding approaches of distribution to handle different types of component of in "Query Processing" unit in Figure 4.6. The procedure to compute the components for different approaches of distribution is completed by processing the ontology metadata. The "Query Distribution" unit on node 1 concurrently distributes the queries to the "Query Processing" units of node 2 and 3 for future processing. The system on Node 2 and 3 realize that no future distribution is required, therefore they come to the query answering procedure. Afterwards, in "Query Answering" unit, on the one hand, the system computes answers for coupled source ontologies $\mathcal{S}_2$ and $\mathcal{S}_3$ (i.e., the node 1 loads the TBoxes of the ontologies in and integrates them locally to compute the answer, whereas the ABoxes of ontologies in still reside on the remote nodes). On the other hand, the self- contained ontologies $\mathcal{S}_1$ and $\mathcal{S}_4$ are handled by an absolutely distributed approach (i.e., the node 1 answers the queries for $\mathcal{S}_1$ and node 3 computes result for $\mathcal{S}_4$). After collecting all the results, the system returns it to the user on node 1.

Hybrid Query Answering

The query answering in this infrastructure is an adaptable hybrid system that can automatically decide which distribution approach to adopt. This infrastructure is *hybrid* because we have two types of distributed approaches for self-contained and coupled ontologies, respectively, which are integrated in the one query answering system in an adaptable manner. Driven by the example scenario in Section 4.3.1, we need to define the different components, but first of all, the basic units of $\mathcal{IS}$ are the *partitions* of $\mathcal{IS}$.

Definition 6 (Partition of $\mathcal{IS}$) *Given $\mathcal{IS} = (\mathbf{S}, \mathcal{T}, \mathbf{M})$, a* partition $\mathcal{IS}_i$ *is defined as $\mathcal{IS}_i = (\mathcal{S}_i, \mathcal{T}, \mathcal{M}_i)$, where $\mathcal{S}_i \subseteq \mathbf{S}$, $\mathcal{M}_i \subseteq \mathbf{M}$, $\mathcal{S}_i \neq \emptyset$, and $\mathcal{M}_i \neq \emptyset$ such that $\bigcup_{i\in\{1,\ldots,n\}} \mathcal{IS}_i = \mathcal{IS}$ and $\forall\ \mathcal{S}_i : \mathcal{S}_i \cap \mathcal{S}_j = \emptyset$,*

$i, j \in \{1, \ldots, n\}$, $i \neq j$. *Given the query answering function* $\mathsf{Ans}(Q, KB)$. *The query answering over a partition* $\mathcal{IS}_i$ *is defined as:* $\mathsf{Ans}(Q, \mathcal{S}_i \cup \mathcal{T} \cup \mathcal{M}_i)$, *where* $KB = \mathcal{S}_i \cup \mathcal{T} \cup \mathcal{M}_i$.

Let's first discover the cases where the whole set of distributed ontologies in $\mathcal{IS}$ can be split into several fragments based on the existing partitions $\{\mathcal{IS}_1, \ldots, \mathcal{IS}_n\}$. Further optimization can be achieved if it is *known* to system from user-defined metadata that the query can be answered against particular components of $\mathcal{IS}$ independently the two categories of ontologies, where the overall answer can be obtained as a simple union of the individual query answers. We define such a particular partition as a *valid component* of $\mathcal{IS}$.

Definition 7 ($\mathcal{IS}_v$: Valid Component of $\mathcal{IS}$) *Given* $\mathcal{IS}$, *a component of* $\mathcal{IS}$: $\mathcal{IS}_v = \{\mathcal{IS}_1, \ldots, \mathcal{IS}_n\}$ *with a set of partitions* $\mathcal{IS}_i = \{\mathbf{S}_i, \mathcal{T}, \mathbf{M}_i\}$ *with* $\mathbf{S}_i = \{\mathcal{S}_1, \ldots, \mathcal{S}_m\}$ *and* $\mathbf{M}_i = \{\mathcal{M}_1, \ldots, \mathcal{M}_m\}$, *is called* valid *if*

$$\mathsf{Ans}(Q, \mathcal{IS}) = \bigcup_{v \in \{1,\ldots,n\}} \mathsf{Ans}\left(Q, \bigcup_{i \in \{1,\ldots,m\}} (\mathcal{S}_i \cup \mathcal{T} \cup \mathcal{M}_i)\right)$$

where Q *is a conjunctive query.*

Obviously, $\mathcal{IS}$ is the valid component of itself, when $n = 1$. Ideally, the query could be answered independently for each source ontology (i.e., if the component only consist of a single element). This is the case, if the individual source ontologies constitute disparate, unrelated data sets. A typical example would be the integration of student databases of multiple universities that do not overlap. We consider this kind of ontologies are *self- contained*, otherwise, they are *coupled*.

Our assumption in managing distributed ontologies is that users often expect to avoid *unintended* correspondences between ontologies and *unexpected* query answering result. We herein apply ontology metadata [HSH$^+$05], which includes the information about (1) whether it contains mapping and (2) the other ontologies that are defined to be related with a particular ontology by the users, to apply different distribution approaches. In particular, for some large distributed ontology data sets, user are often aware of the correspondences between the data sets and want to overall control the query answering by providing the definition of *self-contained ontology* and *coupled ontology*.

Definition 8 ($\mathcal{IS}_s$: Single-element Component of $\mathcal{IS}$) *A component of* $\mathcal{IS} : \mathcal{IS}_s = (\mathcal{S}_s, \mathcal{T}, \mathcal{M}_s)$ *is a* single-element component *if*

$$\mathsf{Ans}(Q, \mathcal{IS}) = \bigcup_{s \in \{1,\ldots,n\}} \mathsf{Ans}\left(Q, \mathcal{S}_s \cup \mathcal{T} \cup \mathcal{M}_s\right)$$

where $\mathcal{S}_s$ *is a* self- contained ontology *and* Q *is a conjunctive query.*

Another special case of *single-element component* also occurs, if a source ontology does not require a mapping with the target ontology. This is the case, if source and target ontology do not differ in their schema (i.e., the TBox part of ontologies), this component of $\mathcal{IS}$ is a *homogenous component*.

Definition 9 ($\mathcal{IS}_h$: Homogenous Component of $\mathcal{IS}$) *Let* $KB_{\mathcal{T}_s}$ *and* $KB_{\mathcal{T}_t}$ *be the ontology Tboxes of* $\mathcal{S}_h$ *and* $\mathcal{T}$, *respectively. A component of* $\mathcal{IS} : \mathcal{IS}_h$ *is a* homogenous component *iff* $KB_{\mathcal{T}_s} \equiv KB_{\mathcal{T}_t}$. *We have:*

$$\mathsf{Ans}\left(Q, \mathcal{S}_h \cup \mathcal{T} \cup \mathcal{M}_h\right) = \mathsf{Ans}\left(Q, \mathcal{S}_h\right)$$

where $\mathcal{S}_h$ *is a* self-contained ontology *and* Q *is a conjunctive query.*

Please note the notion of *self-contained ontology* and *coupled ontology* does not satisfy when the implicit mappings or other correspondences exist. In our work, we only concern explicit mappings defined in the ontology metadata, because the actual users often want to avoid unwanted results caused by implicit mappings or other correspondences while managing data on hand.

Currently we apply two distribution approaches in our system: one is distributed execution, and the other one is to answering queries using central integration. Concurrent execution is established for both approaches, because not only the distributed tasks can be executed in parallel, the central integration may also consist several *valid components* that can be queried individually. Apparently, accessing self-contained ontologies in $\mathcal{IS}_s$ and $\mathcal{IS}_h$ could be fully distributed because it doesn't need to access other source ontologies. We establish a set of $\mathcal{IS}_v$ to handle the ontologies that are coupled, and for each $\mathcal{IS}_v$, we query over it concurrently in the local node. This hybrid query answering approach is able to be applied to any reasoner that supports query answering over ontologies with mappings. This scenario is commonly used in managing distributed ontologies with large size instance data as users often want to control the overall distributed system by avoiding *unintended* correspondences between ontologies and *unexpected* query answering result.

Algorithms

Given the definitions of three kinds of component of $\mathcal{IS}$ with partitions $\{\mathcal{IS}_1, \ldots, \mathcal{IS}_n\}$, $\mathcal{IS}_v$, $\mathcal{IS}_s$ and $\mathcal{IS}_h$, the components of $\mathcal{IS}$ can be directly achieved offline by processing the metadata of ontologies and does not affect the performance of query answering. Using $\mathcal{IS}$ with its partitions $\{\mathcal{IS}_1, \ldots, \mathcal{IS}_n\}$ as the input, the components can be achieved by the prepossessing procedure (c.f. Algorithm 1) by identifying the categories of the ontologies (i.e., self-contained or coupled ontology).

Algorithm 1 Preprocessing Segmentation

Require: distributed ontology integration system $\mathcal{IS}$ = ($\mathbf{S}$, $\mathcal{T}$, $\mathbf{M}$), and partitions $\{\mathcal{IS}_1, \ldots, \mathcal{IS}_n\}$, coupled ontology set $\mathbf{C}$, self-contained ontology set $\mathbf{S}_s$ and $\mathbf{S}_h$ for $\mathcal{IS}_s$ and $\mathcal{IS}_h$, respectively

1: **for all** $\mathcal{S}_i \in \mathbf{S}$ **do**
2: **if** $\mathcal{S}_i \in \mathbf{S}_s$ **then**
3: get mapping $\mathcal{M}_i$, establish $\mathcal{IS}_{s_i}$
4: **else if** $\mathcal{S}_i \in \mathbf{S}_h$ **then**
5: establish $\mathcal{IS}_{h_i}$
6: **else**
7: put $\mathcal{S}_i$ and all ontologies that are coupled with $\mathcal{S}_i$ to $\mathbf{C}_i$
8: get related mappings of $\mathbf{C}_i$ and put them to $\mathbf{M}_i$, establish $\mathcal{IS}_{v_i}$
9: **end if**
10: **end for**
11: output all $\mathcal{IS}_{v_i}, \mathcal{IS}_{s_i}, \mathcal{IS}_{h_i}$

After getting all the components as the required input, herein we introduce the query answering functions for the three types of component of $\mathcal{IS}$ and their distribution approaches. The query answering function in $\mathcal{IS}_v$, $\mathcal{IS}_s$ and $\mathcal{IS}_h$ are

$$\mathsf{Ans}_v(Q, \mathcal{IS}_v),\ \mathsf{Ans}_s(Q, \mathcal{IS}_s),\ and\ \ \mathsf{Ans}_h(Q, \mathcal{IS}_h)$$

respectively, where Q is a conjunctive query. The computation of Ans_v is concurrently executed for each *valid component* of $\mathcal{IS}$ in the local node. Apparently, the concurrent execution by creating local threads for each $\mathcal{IS}_v$ reduces potential overheads caused by the network imbalance in accessing remote $\mathcal{IS}_v$. The answers to the queries is integrated after all the threads return the answers to the system. In the distribution approach for $\mathcal{IS}_s$ and $\mathcal{IS}_h$, the local node sends the queries to $N_i = \mathsf{Loc}(\mathcal{S}_i)$ in the distributed network in a concurrent manner. To be specific, one thread sends the query to a remote node where the source ontology resides in the distributed network, together with other threads in a concurrent way.

After getting the three different kinds of component and corresponding ontologies (i.e., coupled ontologies in $\mathcal{IS}_v$, self-contained ontologies in $\mathcal{IS}_s$ and self-contained ontologies in $\mathcal{IS}_h$), the system distributes the queries in the network and compute the result.

The Algorithm 2 provides efficient query distribution and answering over networked ontologies in the distributed environment. In Algorithm 2, each component of $\mathcal{IS}$ is treated in different manners based on their

Algorithm 2 Query Distribution and Answering Algorithm

Require: a conjunctive query Q, components $\mathcal{IS}_{v_i}$, $\mathcal{IS}_{s_i}$ and $\mathcal{IS}_{h_i}$

1: **for all** $\mathcal{IS}_{v_i}$ **do**
2: compute $\text{Ans}_v(Q, \mathcal{IS}_{v_i})$ concurrently on local node
3: **end for**
4: **for all** $\mathcal{IS}_{s_i}$ **do**
5: get source ontology $\mathcal{S}_i$ of $\mathcal{IS}_{s_i}$
6: get remote node $N_i = \text{Loc}(\mathcal{S}_i)$ in $\mathcal{IS}_{s_i}$
7: compute $\text{Ans}_s(Q, \mathcal{IS}_s)$ on node N_i concurrently in the distributed network
8: **end for**
9: **for all** $\mathcal{IS}_{h_i}$ **do**
10: get source ontology $\mathcal{S}_i$ of $\mathcal{IS}_{h_i}$
11: get remote node $N_i = \text{Loc}(\mathcal{S}_i)$ in $\mathcal{IS}_{h_i}$
12: compute $\text{Ans}_h(Q, \mathcal{IS}_h)$ on node N_i concurrently in the distributed network
13: **end for**
14: results collection and display

characters: The local node executes Ans_v for each individual $\mathcal{IS}_v$ concurrently to compute the result; for $\mathcal{IS}_s$ and $\mathcal{IS}_h$,query Q is distributed to node N_j by executing the functions Ans_s and Ans_h in a concurrent way. Finally the results are collected and returned to the user.

The complexity of Algorithm 2 is decided by the complexity of ABox reasoning [HMS05] that is NP-COMPLETE in worse case. Furthermore, the complexity of query language is also a desiderata in actual applications, for example, the complexity of SPARQL query language is NP-SPACE [PAG06].

4.3.2 Implementation of KAONWeb

Our distributed query answering infrastructure is implemented in a web-based networked ontology query answering system called KAONWeb. KAONWeb has three major components: (1) publishing and reuse component for users identifying and defining information about ontologies, (2) a metadata registry processing ontology metadata, (3) distributed query answering for networked ontologies. The KAONWeb systems are deployed on a set of distributed nodes that both publish their services and consume services of other nodes (c.f., Figure 4.7). Nodes are autonomous and distributed, they work as both servers and clients to the other nodes. In our approach, nodes discovery is achieved similar to the peer discovery mechanism introduced in Oyster system [PH05b]. We introduce the structure of this system in detail below.

The first one is the *Ontology Publishing Service* that connects to the local ontology repository and the local administrator can choose ontologies to be published. Published ontologies will be collected into a temporary repository and ready to be sent. Remote users can then discover the accessible resources on this node and invoke the service to access the ontologies. The relevant ontology information is recorded with the node's IP address as an entry in "Remote Ontology Info Repository" in Figure 4.7.

The second one is the *Metadata Registry Service*, which takes the responsibility of local ontology metadata management, is a reengineering of metadata registry in KAONp2p system [HW07]. The ontology mapping metadata are used to register those ontologies that hold mappings between two ontologies. Such information is stored in the metadata registry. As described in Section 4.3.1, the metadata information is a key issue in partitioning the $\mathcal{IS}$ in an appropriate way. If an ontology is published, its associated metadata are also discoverable to the remote users. This procedure is depicted as the second item of "Web Service Consumer" box in Figure 4.7. Moreover, a "Virtual Ontology" repository is automatically created with target ontology $\mathcal{T}$ in $\mathcal{IS}$.

The last but the most important one is the *Query Answering Server*. "Remote Ontology Info Registry" is a structured information container with the remote nodes' IP addresses and a list of their published ontologies'

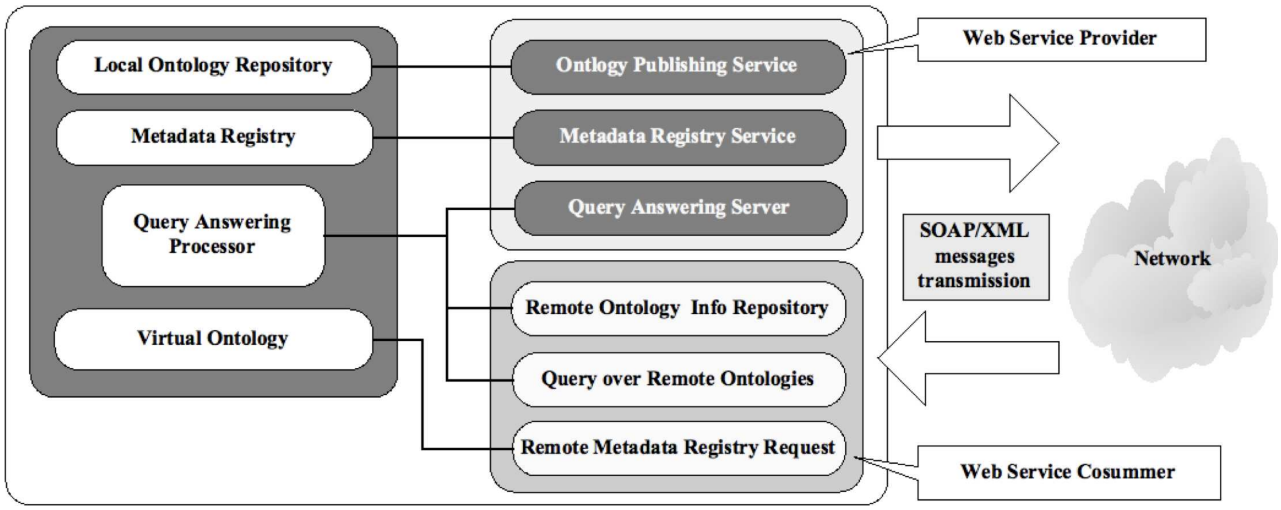

Figure 4.7: Web Service structure in KAONWeb

URIs. When a query is requested by a user, the system in this user's node distributes the query to the remote nodes where the ontologies are selected to be queried over, then the "Query Answering Server" on these remote nodes execute the query and send back the results. According to the study in [MS06], KAON2 has comparatively better query answering performance over instance data than other reasoners, thus, we've chosen KAON2 as the background query answering engine in the KAONWeb implementation. This implies we are using $\mathcal{SHIN}(\mathbf{D})$ as the ontology representation formalism[MS06], whereas this infrastructure can be generally implemented and deployed by support from *any* OWL-DL reasoner that provides server or service for query answering over ontologies.

4.3.3 Evaluation

The evaluation of the novel query answering algorithm implemented in the KAONWeb system is introduced in this section. First we elaborate the evaluation settings that include evaluation purpose, infrastructure, data and criteria; then we introduce the two experiments in detail and finally discuss the result.

The assumption for our system is that this approach is able to scale over a large number of nodes regardless the possible network overheads. We realized that our experiments have been done in a practical networking environment that all the distributed computer are connected by internet. Therefore we expect the when the number of distributed nodes scale up, there will a significant network overhead.

Settings

We evaluated the performance of the proposed algorithms against the centralized query answering performance. There were two experiments for this evaluation: One was to test the performance when the ontology network consisted of half self-contained ontologies and half coupled ontologies; the other one was to test how different degrees of coupling affect the performance of the entire system.

We set up the evaluation environment in the following way: We had 8 computers, in which there were 2 workstations and 6 personal computers distributed in the internet, to simulate a practical distributed networking scenario. Each computer played the role as a distributed node and held a data set with fixed size. The first experiment aimed to compare the performance owned by distributed system against the centralized query answering system. The second experiment measured the performance influenced by the degree of coupling, where only a distributed scenario was required.

The data sets were Lehigh University Benchmark (LUBM) [GPH05] with LUBM automatically generated

ontologies that includes instance data of information about university life, approximately 9MB of OWL/ RDF data, and SWRC ontology. (`http://ontoware.org/projects/swrc/`) Each node held a LUBM data set and either SWRC schema or LUBM schema. The performance affected by mappings was not the concern of this evaluation because it had been studied in [HW07]. In this evaluation, the coupling between the data sets was defined by the user. For example, if we aligned this data setting to Figure 4.6 as a three distributed nodes case, $\mathcal{T}$ was an arbitrary defined university ontology with LUBM schema; $\mathcal{S}_1$ was a self-contained ontology in $\mathcal{IS}_h$ with LUBM schema and data set; $\mathcal{S}_2$ and $\mathcal{S}_3$, which are ontologies with SWRC schema, were defined as coupled with each other in $\mathcal{IS}_v$ (presented as dashed line) and they were mapped to $\mathcal{T}$ by via $\mathcal{M}_3$ and $\mathcal{M}_1$, respectively; $\mathcal{S}_4$, which is an ontology with SWRC schema and data set in $\mathcal{IS}_s$, could be mapped to $\mathcal{T}$ via $\mathcal{M}_2$; $\mathcal{M}_1$, $\mathcal{M}_2$ and $\mathcal{M}_3$ contained mappings between SWRC and LUBM schema with different contents. We still used the same query in the KAONp2p evaluation (c.f. Section 4.2.4). An experiment unit was one execution of one query over a certain distribution setting (i.e., the execution of Q_i on n distributed nodes, where $i = \langle 1, 2, 3\rangle, n = \langle 2, 4, 6, 8\rangle$). The three queries above were executed for 20 times for each experiment unit and the average execution time T_d was computed, recorded and compared with the execution time held by centralized query answering over same size of data T_c. The performance increase/decrease of distributed setting with respect to the centralized setting was computed by the equation: $\frac{T_c - T_d}{T_d}$.

Experiments and Analyses

Performance and Scalability First of all we needed to compute the performance of centralized system. We resided all the distributed data in a local node and used reasoner to query all the ontologies in a localized way (e.g. If we query the ontologies on 8 distributed nodes with 1 data set on each of them, then we compare the performance with the query answering performance owned by local reasoner with same 8 data sets). Then we deployed 8 physically distributed nodes for this experiment to compute the performance of distributed system. There were four self-contained nodes and four coupled nodes defined by the user, in which the data sets on self-contained nodes can be directly queried in a complete distributed manner, whereas those on coupled nodes needed to be processed in ontology integration system. We started the experiment by executing the queries above on two distributed nodes, then four, six and ended up with eight nodes, in which consisted always half self-contained ontology(ies) and half coupled ontology (ies). After setting up the experiment environment, we executed the queries introduced in Section 4.3.3 distributively.

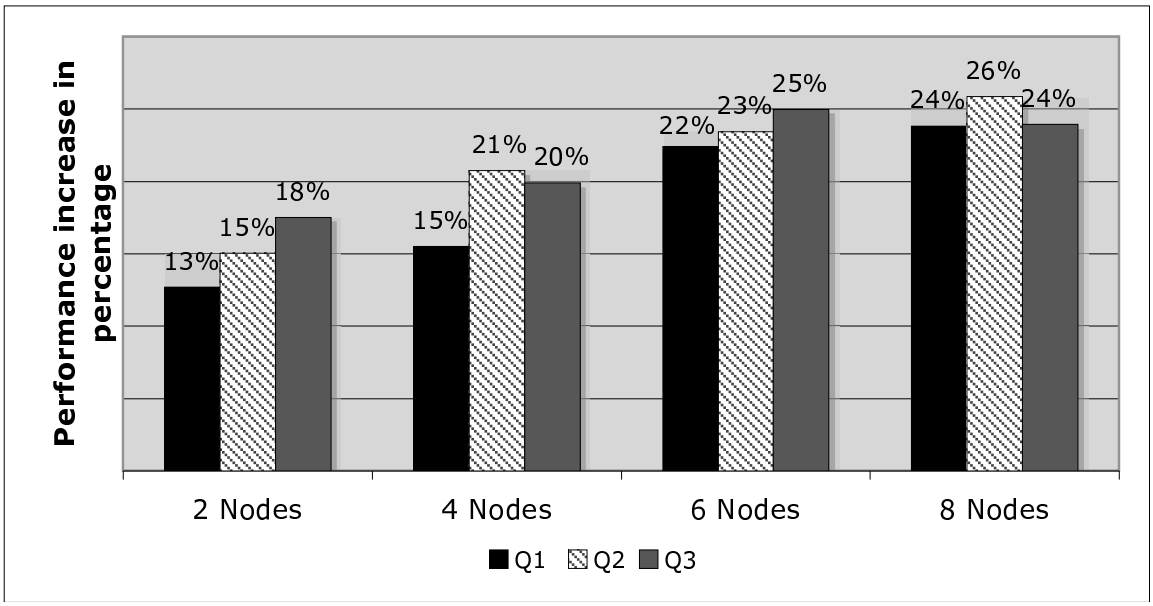

Figure 4.8: Distributed query answering performance *increase* against centralized query answering.

Compared to the centralized query answering, the performance of distributed execution is considerably better when the ontology network that consists of half self-contained ontologies and half coupled ontologies. Moreover, it is not difficult to discover that when the number of distributed nodes scales up, the performance increases are more stable, which indicates the good scalability of the algorithms. However, we also see a remarkable network overheads that causes the performance overheads. For example, the 8 nodes situation is not much better than 4 nodes.

Impact from Degrees of Coupling In the second experiment, we wanted to discover how the degrees of ontologies coupling affect the overall performance, therefore we simply executed the queries on the four distributed nodes with self- contained ontologies only and coupled ontologies only, respectively. We collect and compared the result with those held by distributed execution on four nodes in the first experiment in Figure 4.9.

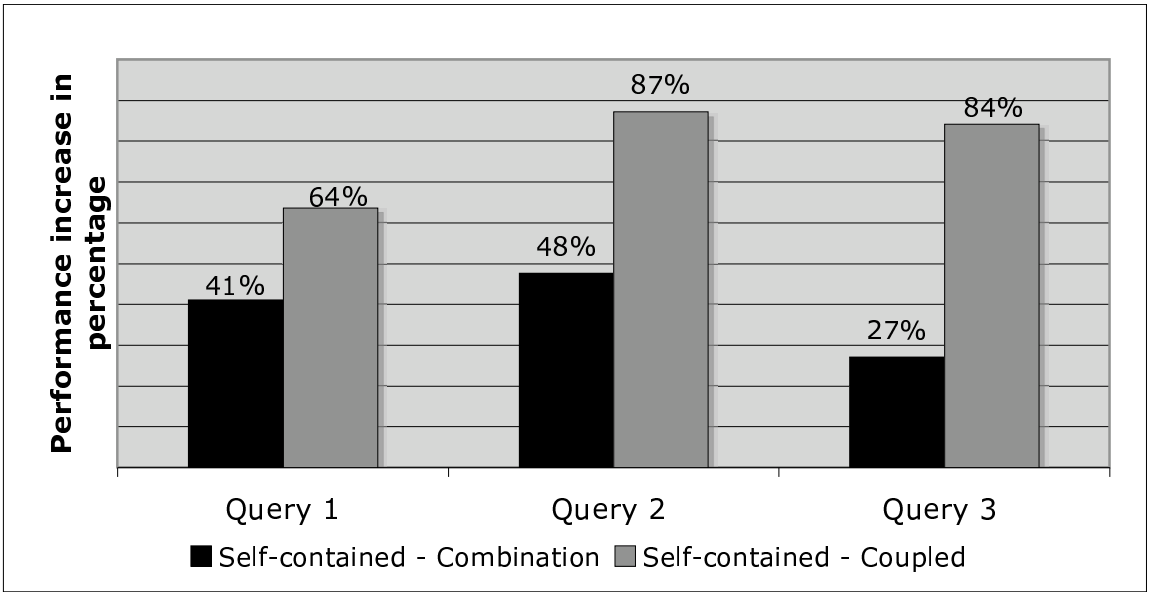

Figure 4.9: The performance *increase* with respect to the degree of coupling.

The Figure 4.9 depicts the result of the second experiment. It indicates that the efficiency of our algorithms is affected by the degree of ontologies coupling in the network – the less coupled ontologies exist in the network, the better performance the system has. Further, in the worst case – the ontologies in the network are all coupled with each other, the distributed query answering is still similar to the centralized one using local reasoner.

Analyses In the ideal case, there is more performance increase when the number of nodes scales up. Base on our algorithm, taking 8 nodes distribution for example, assume we query 8 sets of data locally in serial, the query answering time is T. We also assume an ideal environment for the first experiment: There is no network overheads, all the nodes have the same computing capacity, and the node where the query is executed contains coupled ontology. We can easily get the ideal execution time: $\frac{T}{2} + \frac{T}{8} = \frac{(5 \cdot T)}{8}$ (The node where the query is executed processes 4 sets of data in $\frac{T}{2}$ and each concurrent node processes 1 set of data in $\frac{T}{8}$). We therefore have the performance increase of 37.5% in this ideal case. We conclude that the network overheads and the imbalance of computing capacity of different distributed nodes leads to the performance overheads in the practical experiments.

4.3.4 Conclusions and Future Work

In this section, we introduced an optimized distributed infrastructure for networked ontology query answering. We deployed this infrastructure in our KAONWeb system that consists of query answering and distributed

metadata registry components to handle distributed ontologies in a networked scenario. Novel algorithms for distributed ontology query answering were implemented and evaluated. The evaluation showed that our novel algorithms took advantage of the hybrid approach that consisted of both task distribution and centralized integration, and it had remarkably better performance compared to the centralized query answering. Moreover, it scaled well within such a networked and distributed ontology scenario in an adaptable manner. We argue this scenario is common in the today's Semantic Web and expected to be more popular with the progress of many related mainstream projects concerning query answering in semantic data integration (e.g. NeOn project [DMS+05] and DartGrid Traditional Chinese Medicine project [CWW+06]).

There are two dimensions for future work. (1) The networked ontology consists not only the mapping information, but modularity information is also important for distributed processing. With the future development of networked ontology model in NeOn project, the system should be able to handle distributed modular ontologies. (2) The coupled distributed ontology TBoxes are able to be effectively handled in a complete distributed manner by formalisms such as Distributed Description Logics [BS03]. We plan to optimize our algorithms if both TBox and ABox are effectively supported by these formalisms in the future.

Chapter 5

Conclusion

5.1 Summary

Next generation semantic applications will be characterized by a large number of networked ontologies in a distributed networking scenario. In this deliverable we addressed this challenge by proposing a novel integrated architecture to support the development life-cycle of networked ontologies in the NeOn project. We also considered the requirements from case studies Workpackage 7 and 8, and used the NeOn general architecture provided by Workpackage 6.

Given the foundations for managing networked ontologies, we introduced the components in this integrated architecture listed below:

- The Oyster distributed ontology metadata registry;
- the KAONp2p decentralized query answering infrastructure focusing on ontology sharing with high autonomy;
- the KAONWeb, an distributed ontology query answering system with particular optimization for distributed networks.

Oyster aims to provide the metadata registry to help in managing distributed ontologies. The new development of Oyster featured in providing a set of well-defined APIs and web services, as well as support of SPARQL query language and KAON2 reasoner.

The major difference between the KAONp2p and KAONWeb was the application domain: As Peer-to-Peer computing differed from classic distributed computing, we provided two solutions to query answering over distributed ontologies.

KAONp2p was designed for sharing ontologies that were created in the decentralized setting, the central contribution discovered in the evaluation of KAONp2p was that KAONp2p had similar performance with centralized query answering although the ontologies were distributed.

Because KAONp2p was not designed for querying ontologies that were centrally created at first and distributed afterwards, we implemented KAONWeb as an alternative system that was particularly optimized in distributively processing query answering tasks over distributed ontologies.

We evaluated our query answering component and the evaluation results showed that our prototypes were very applicable to handle the networked ontologies in a efficient way. However, there were certainly several directions for further optimization and improvement.

To facilitate the usage of the systems introduced in this deliverable, we also provided the software instructions as appendix in the end of this deliverable. The instructions taught readers how to download and setup the software systems, and illustrated some comprehensive use cases to help in understanding the central functionalities of the systems provided here.

5.2 Future Roadmap

The specification of this integrated architecture is only an initial step of establishing a final reference architecture for networked ontology management in the complex networking scenario. This effort is going to be tightly aligned to the NeOn architecture defined in the Workpackage 6. Therefore we discuss the future steps in the following. The roadmap that we are going to follow as the future work include:

- A unified system for networked ontology management, including the APIs for managing networked ontologies with different algorithms for distributed querying answering, is planned for the next step.
- Extensions and refinements of the current ontology metadata and their supporting platform to handle other aspects of networked ontologies, such as an explicit representation of ontology modularity via metadata and provide distributed reasoning functionality to support modular ontologies.
- Optimization for reasoning and query answering based on the analyzing the ontology metadata. For example, if an ontology's metadata state that this particular ontology is not going to be associated with other ontologies, the query answering task then can be direct executed over this ontology without considering integrating with other ontologies.
- Optimization of query answering over distributed ontologies considering different semantics in distributed networks. Advanced and efficient query answering techniques such as approximate query answering is also an important issue to be addressed.
- Enhance the versatility of network ontology usages for management of other formats of semantic data, such as relational data and XML data.

The extensions and refinements will be the focus of the work on the subsequent Deliverable D1.4.2 *Prototypes of Networked Ontology Management, Updated Version*. In particular, we will develop the metadata for ontology modularity to support the reasoning over modular ontologies. At the same time, we will consider the optimizing for distributed ontologies with focus on the analysis of semantics of ontology mappings. Moreover, supporting multiple formats of data described by networked ontologies is also one of our major concerns in networked ontology management.

Appendix A

Appendix: Software User Instructions

In this appendix, we provide the detailed information for the components of integrated architecture for managing networked ontologies, including the instructions for downloading, installing and using of these software prototypes. All the prototypes are implemented in Java 1.5, so please download the latest JDK 5.0 (available at: `http://java.sun.com/j2se/1.5.0/download.jsp`) for running the programs.

A.1 Oyster User Instructions

A.1.1 Download and Setup

Oyster is supported in two operating systems, i.e. Windows and linux. We are working on the Macintosh release.

Windows version: Download the latest release for windows (oyster2-win32-installxxx.jar) from Oyster web site (`http://oyster.ontoware.org`), execute the file and proceed as follow:

- Click next on the welcome window
- Select *Autostart Oyster on system startup* check box if you want to start Oyster automatically every time you start windows
- select the installation path
- click next when the *Overall installation progress* bar completes
- Select a program group for the shortcuts and the name for the new program shortcut
- finish the process clicking on done

To uninstall Oyster, execute the file *uninstaller.jar* that was generated in the directory *programfiles/Oyster application/unistaller*.

Linux version: Download the latest release for linux (Oyster2-xxx-linuxx.zip) from Oyster web site (`http://oyster.ontoware.org`), unzip the file within the directory you want to install Oyster and run the script *start.sh*.

To uninstall Oyster, just delete the directory where Oyster is installed.

After the installation of Oyster, the first time the system is executed, a dialog window asks the user to enter (or accept the default values) the following configuration parameters:

- File location of the Local Expertise Registry Ontology. This file keeps the local node repository, which stores the peer knowledge (e.g. local ontology metadata, known peers along with their expertise, etc.).

- File location of the Peer Description Ontology. This ontology is the OMV extension for the peer metadata.
- File location of the OMV Ontology.
- File location of the Topic Ontology. This ontology is the topic hierarchy that will be used as the range for the OMV property *domain* (i.e. DMOZ, ACM, etc.).
- File location of the Oyster logo image. It has to be an image file (e.g. jpg, bmp, etc.).
- The root concept of the OMV Ontology (i.e. ontology).
- The root concept of the Topic Ontology.
- Ten search conditions (i.e. OMV properties). These properties allows to user to specify conditions when querying the registry.
- Peer Name. The name of the Peer should be at least 4 characters long. The name is not necessary to be unique in the network, since internally the system uses a GUID, however it is recommended that the name reflects some useful information i.e. name of the user+organisation.
- Peer Type. The peer type can be either *R* if the peer is a Rendezvous Peer, or *S* if it is a Simple Peer.
- Bootstrap Peer Name and IP address. It is the name and IP address (respectively) of a Rendezvous Peer that will be contacted by the local Peer when it starts to announce its presence and collect information of other peers in the network.

All of these configuration parameters can be changed/updated later using the preferences menu (see next section), except the peer name and type.

A.1.2 Usage

Querying

Scope: You can select the scope of your query:

- Local Peer: Restrict your query to your local peer.
- Automatic Search: Intelligent selection of peers.
- Selected Peers: Select a set of peers.

Search details: You can restrict your query to special attributes (e.g. search for ontologies in OWL), or for ontologies about a specific subject (based on the classification ontology loaded, such as DMOZ), or you can search for a string match in any attribute. Furthermore, you can search ontologies by their namespace (uri), and the result will show all the ontologies that includes the words on the uri field on their namespace.

Save items

You can save items to your local node repository by simply selecting the item and pressing the "Save" button.

Adjusting the columns

By pressing the right mouse button inside the result view panel you can adjust your personal view (see figure A.1).

Create and Edit items

New ontology metadata entries can be created from scratch, or existing ones can be modified. Oyster provides two templates for default to create a new metadata entry from scratch: OntologyFull which includes all the attributes proposed in OMV, and OntologyRequired which includes only the attributes the OMV propose

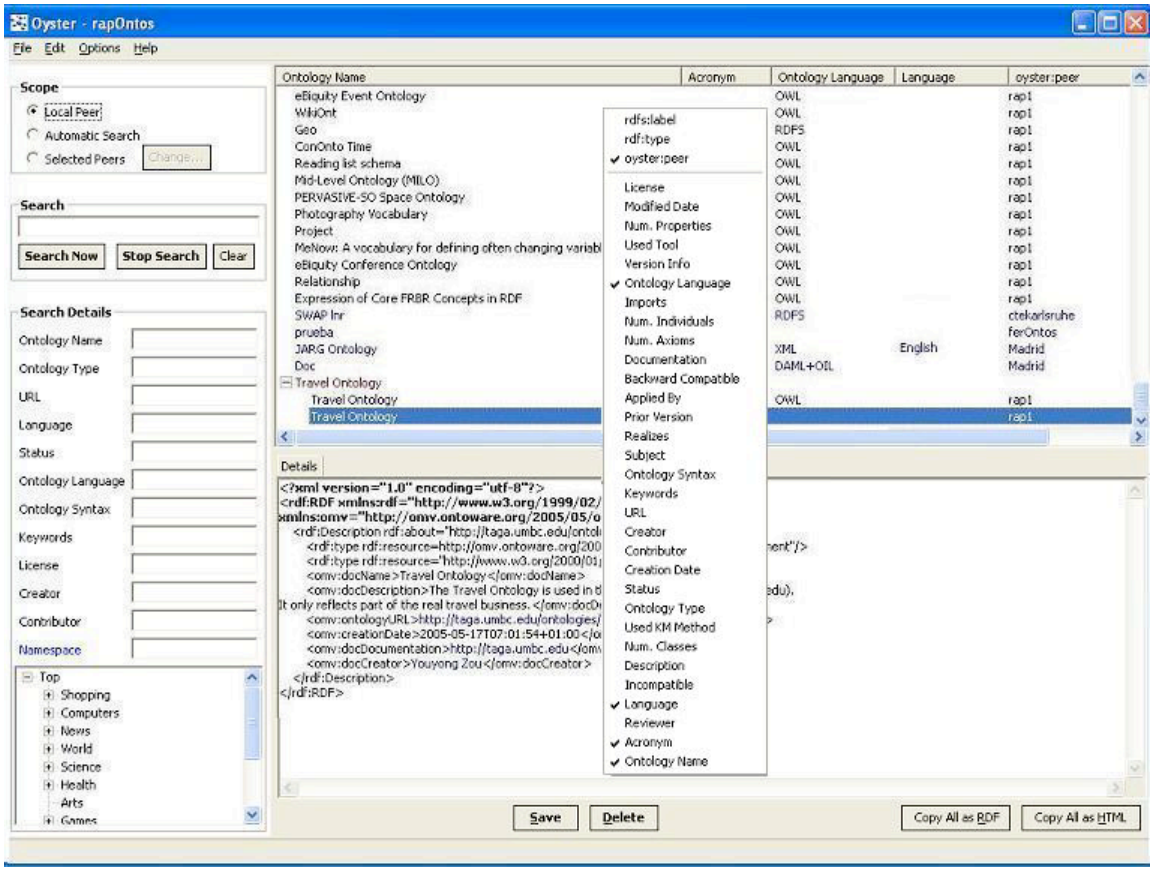

Figure A.1: Personalizing columns

as required. You can also create your own templates or you can create the metadata entry without using templates and just adding the attributes you want to include. It is important to notice that the name and uri properties are mandatory in order to create a new metadata entry.

Import functions

It is possible to import an ontology file document to extract the ontology metadata and then fill missing values (see figure A.2). Oyster supports the OWL, DAML+OIL and RDF-S ontology languages. Also, a complete repository (a RDF file) can be imported into the local repository.

Export repository

The repository can be exported to a HTML file (based on a configurable XSLT file) or to a RDF file. It is also possible to export individual entries.

Preferences

Using this menu, the user can change the configuration parameters given the first time Oyster was executed. For instance, the classification ontology loaded by default is based on DMOZ, but it is possible to load different classification ontologies by changing which file to open in the preferences menu. You can also change the location of the local repository, the templates, OMV ontology or the search conditions.

Duplicate detection

The duplicate detection function, group Ontologies detected as duplicates. Two ontologies are considered duplicates if they have different uris but they realise the same conceptualisation (see section 4.1.2).

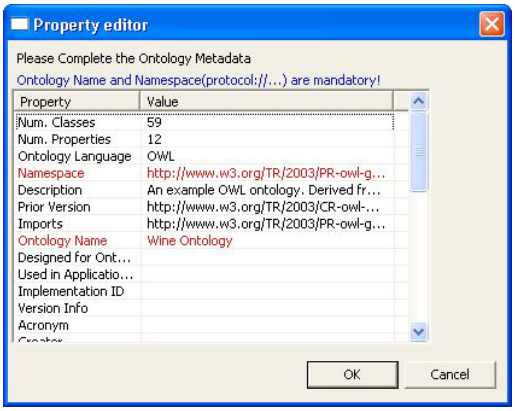

Figure A.2: Oyster import result

A.2 KOANp2p User Instructions

A.2.1 Download and Setup

The initial setup is not straightforward because this system are being developed and tested under a specific Java software developing environment Eclipse, in which developers are able to simply design software user interface by generating GUI forms. Furthermore, many third-party Application Program Interfaces (APIs) are loaded to minimise the programming task, thereby we also need to install those packages in local paths before compilation. There is a released package that supports Microsoft Windows operating system is available to download: `http://kaonp2p.ontoware.org/KAONp2p_install_win32_040.zip`. After downloading and unpacking the KAONp2p release file, we can find a property file of the system and an initial configuration is required for the property file. In the property file, we list all the ontologies required to start the application. The user must configure them according to his file directory and specify the domain ontologies he wants to use for the application. For the detail configuration step, please refer to the readme file in the released package.

For the first time of start, if the local expertise registry is empty, a user is required to input a user name for naming the local peer and needs to indicate the peer type, namely, simple peer or bootstrap peer. The user needs also to indicate some bootstrap peers (at least one bootstrap peers) with known IP addresses such that the new joining peer can perform the peer discovery process and join the P2P network.

A.2.2 Usage

In the release package of the software, we can find a $kaon2.jar$ file. As we have introduced, KAONp2p depends on the KAON2 ontology management system and uses KAON2 API for the management of ontology resources. A KAON2 instance should also be started on each peer before starting the KAONp2p application. The KAON2 instance and the application can be started by the following command:

```
java  KaonP2P.Start  ontologyDirectory
```

The “ontologyDirectory” is the path of the folder where you store your shared ontology documents. After the software started, the user can import new ontology documents that he wants to share with other peers from the ontologyDirectory and the metadata of the new ontology documents will be stored to the local expertise registry. Like this, when the system performs periodical exchange process, other peers in the peer community will be aware of the change about the available knowledge in the network and update their own local expertise registry. Note that, the ontology URI and document name and other important properties (like “imports”) of the imported ontology document will be automatically extracted by the system and the user must specify the expertise domain of the ontology being shared such that it can be classified in the local expertise

registry according to the classification of the topic ontology used in the system (e.g., ACM ontology). The ontology URL and its expertise domain are mandatory for the classification. In the Figure A.3, we give an idea about the "import" property editor.

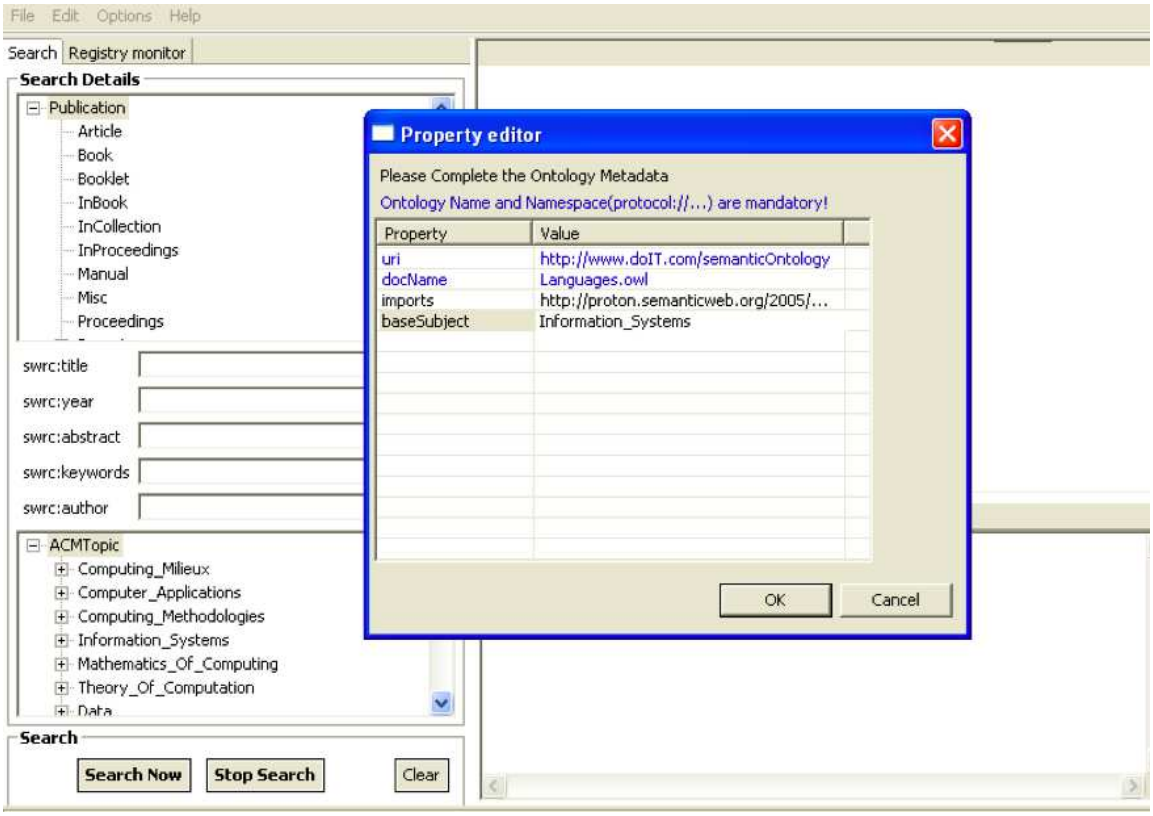

Figure A.3: KAONp2p runtime snapshot.

In the following, we introduce three use cases to show how KAONp2p works as basic usage examples.

Searching and Integration for Shared Resources

In this case, all three peers use the same domain ontologies SWRC and ACMTopic ontology, therefore no mapping is needed between the domain ontologies. We search for bibliographic data – all the Publication data which have expertise topic: Hardware. The results can be illustrated in the figure below. As we can see, all the resources satisfying the query requirement are found out and the metadata of the resources are demonstrated in the detail viewer. If we add a restriction condition to filter out the books which are published on 1999, the results are well filtered by using the restriction condition year: 1999. The relevant result is shown in Figure A.4.

From the figure above, three bibliographical data instances are extracted from two different ontology documents (from the virtual ontology generated by the system, we can see which ontologies are imported to achieve the searching process), the system integrates the data and returns them back. The data have respectively the types: InProceedings and Book which are subclass of Publication.

Mapping between Different Domain Ontologies

In this test case, we will test the effect of mapping mechanism. The two peers located on local host use Proton ontology as the domain ontology and the remote peer use SWRC ontology. All of them use ACMTopic ontology as the topic hierarchy ontology. We assume a rule in the mapping ontology: the Publication in

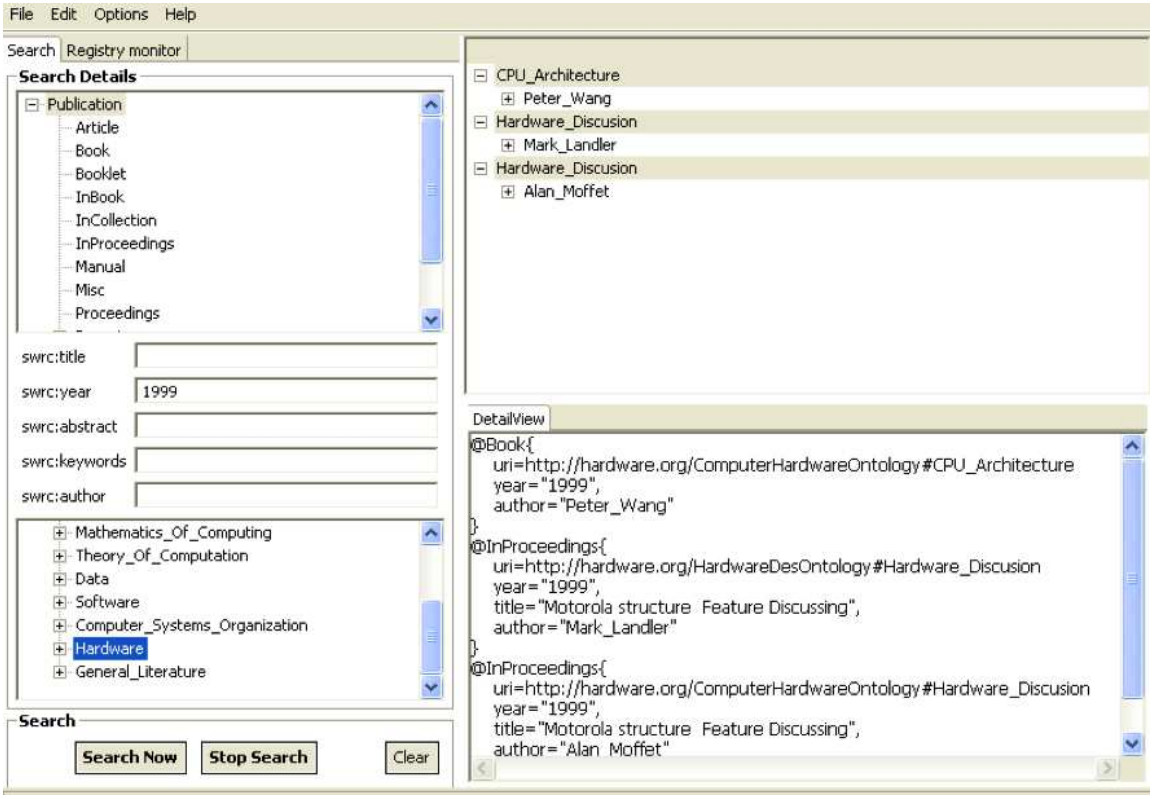

Figure A.4: Searching and integration for shared resources.

SWRC has the same meaning as the Document class in Proton. We issue a query from local host to search for all the publication data. As we can see, the data instances of type Publication being stored on the remote host are extracted as well. And from the illustration of the virtual ontology, we can find the mapping has been imported in the virtual ontology. Additionally, we define another rule in the mapping ontology: The property swrc:year in SWRC has a relevant mapping with the protons:hasDate property of Proton. If we query the Document data which hasDate on 2004, the data instances of Publication which are published on 2004 are also retrieved.

Combination of Ontologies to Extract Resources

We have mapping ontology that has been imported to across the different domain ontologies used by the peer community, resources data could still not be retrieved from single ontology, at this time the combination of ontologies is necessary. We will give a sample test case to illustrate the reasoning ability of KAONp2p illustrated in Figure A.6. For example, in ontology A, it contains some data instances such as: Publication "Consistent Evolution of OWL Ontologies" is publishedAt the conference ESWC. In ontology B, we defined a restriction for the property publishedAt, its domain is limited as InProceedings and its range is Conference. This restriction means that all the Publication data which are publishedAt a conference is a data instance of InProceedings. If a user query for the bibliographical data instances of InProceedings, neither of the above ontologies can answer the query. If we combine them together (import them into the virtual ontology) and with the help of reasoning ability of KAONp2p, the system will consider "Consistent Evolution of OWL Ontologies" as a data instance of InProceedings and return it back.

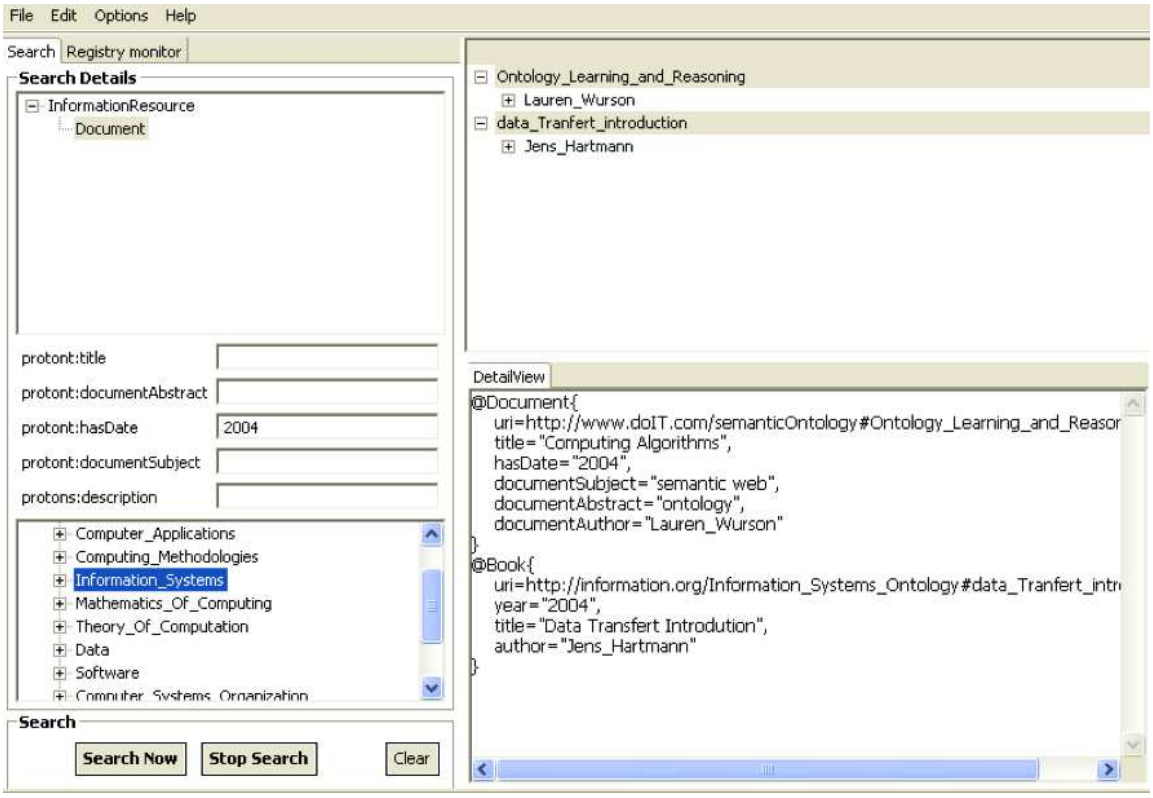

Figure A.5: Ontology mappings between different domain ontologies.

A.3 KAONWeb User Instructions

A.3.1 Download and Setup

The current version of KAONWeb can be downloaded from: `http://www.aifb.uni-karlsruhe.de/WBS/ywa/kaonweb/KAONWeb.zip`. KAONWeb is a web application that can be easily accessed via popular web browsers. There are three step of installation:

1. Before installing KAONWeb, users must first install the Apache Tomcat 5.x (You can find it at: `http://tomcat.apache.org/`) or higher version and make sure it works finely. The Port should be set as "8080".
2. Deploy the $KAONWeb\text{-}ref.war$ package into "$TOMCAT_HOME$/webapps" and then turn on Tomcat.
3. Now it can be accessed under the address `http://localhost:8080/KAONWeb-ref` in the browser.

A.3.2 Usage

Here we provide the general user guide for the first run of KAONWeb:

1. Before using this application, users should put all local OWL files into the folder "$TOMCAT_HOME$/webapps/KAONWeb-ref/ontologies/kaon2server_root".
2. Under the "Catalogue" at the left side of page, there are two functional components named as "Publishing and Reuse" and "Query Answering". We can use the first component for ontology and metadata management, and the second component for query answering.

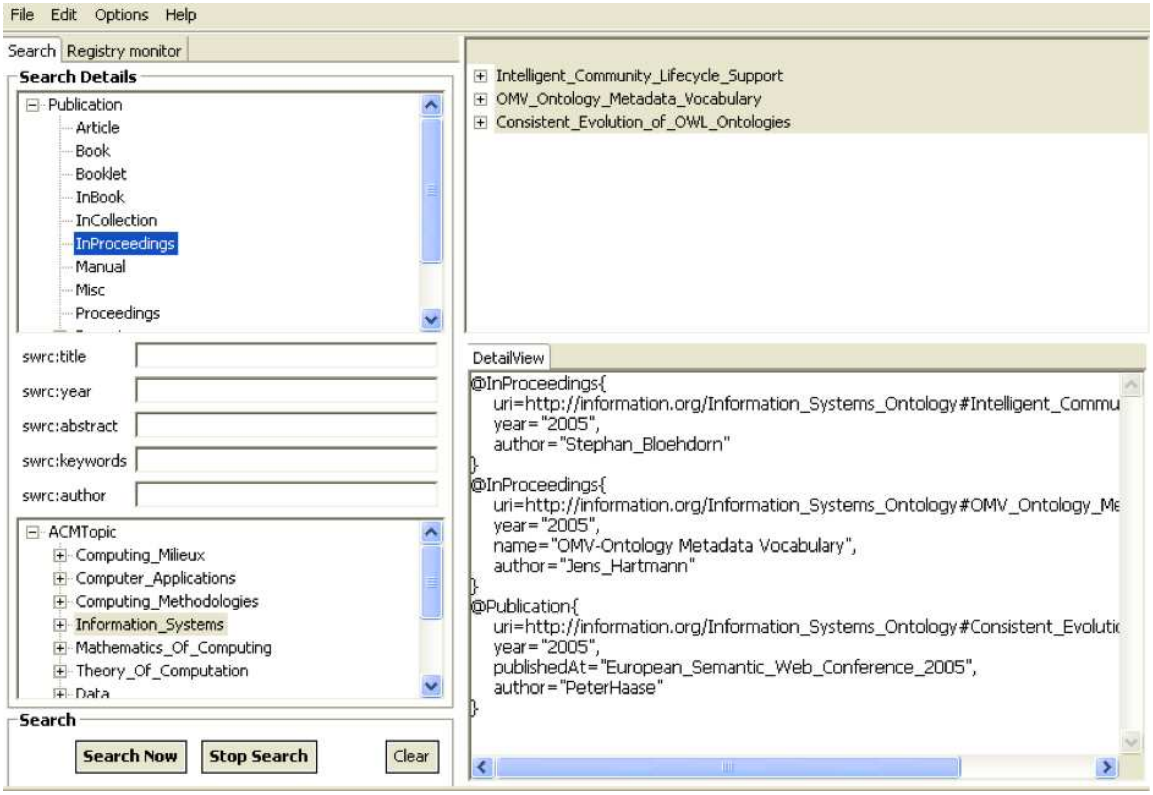

Figure A.6: Combination of ontologies to extract the relevant resources.

3. Now, click the “Management” in “Catalogue” and go into management page. In the “KAON2 Server Status” box, click the switch to turn on the KAON2 server. In the “Ontology Management”, all the local ontologies have been initially listed, and users can also type in the remote peer IP with its web server PORT, which likes the form of “127.0.0.1:8080”, and the list of ontologies will be shown in a new tab. Users can also edit the metadata for each local ontology but not the remote.

4. User can choose an ontology for querying by clickung the button “Set This Ontology For Query”.

5. Click the “Query Answering” button, user can type in the SPARQL query language string. After clicking the “Query” button, all results will be listed out at below.

We also outline two specific use cases in order to provide progressive understanding. In this chapter, there will be a series of snapshot for our activities. These use cases are based upon the example ontologies provided in the software package.

Publishing and Reuse

We first look at the “Publishing and Reusing” page and publish some local ontologies (two autonomous ontologies $ComputerHardwareOntology.owl$ and $informationSystem.owl$ and one mapping $swrc\_proton.owl$), which associates also the metadata information (here is mapping information). Then we type the IP address of the host computer to make a remote access. Please note that this is the same way the nodes can also achieve over the internet. The published ontologies and the mapping set just now are shown on the remote side. The two processes have been shown in Figure A.7.

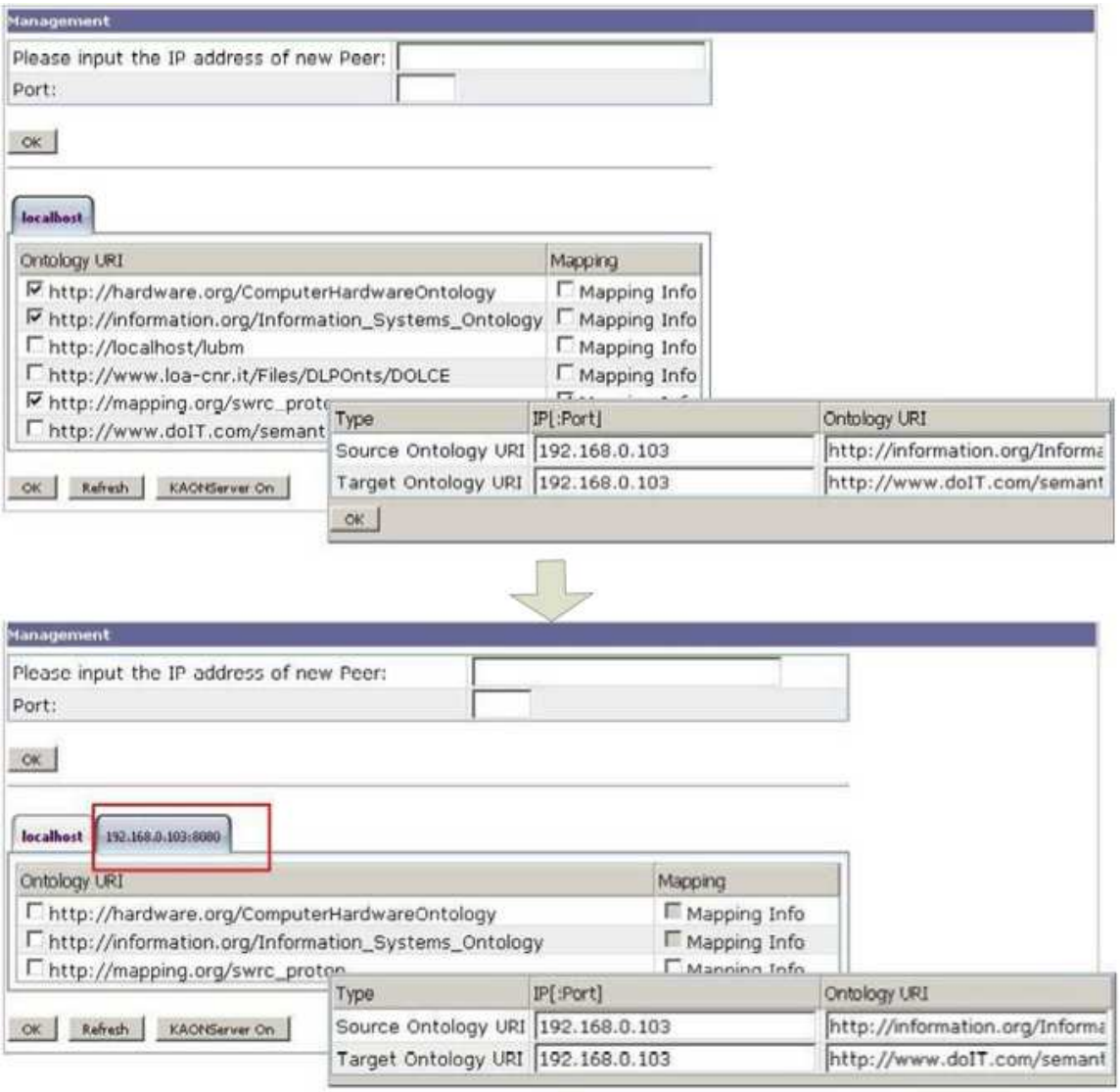

Figure A.7: Publishing and reuse component use case example.

Query Answering

We can perform a sample query upon a hybrid set involving an autonomous ontology (the $ComputerHardwareOntology.owl$) and an integrated system (the mapping $swrc\_proton.owl$, its source ontology $informationSystem.owl$ and its target ontology $Languages.owl$). The query is:

```
SELECT ?x WHERE
{ ?x rdf:type <http://swrc.ontoware.org/ontology# Publication> }
```

Results are outlined in Figure A.8, comprising entries from the autonomous ontology and entries from the integrated system.

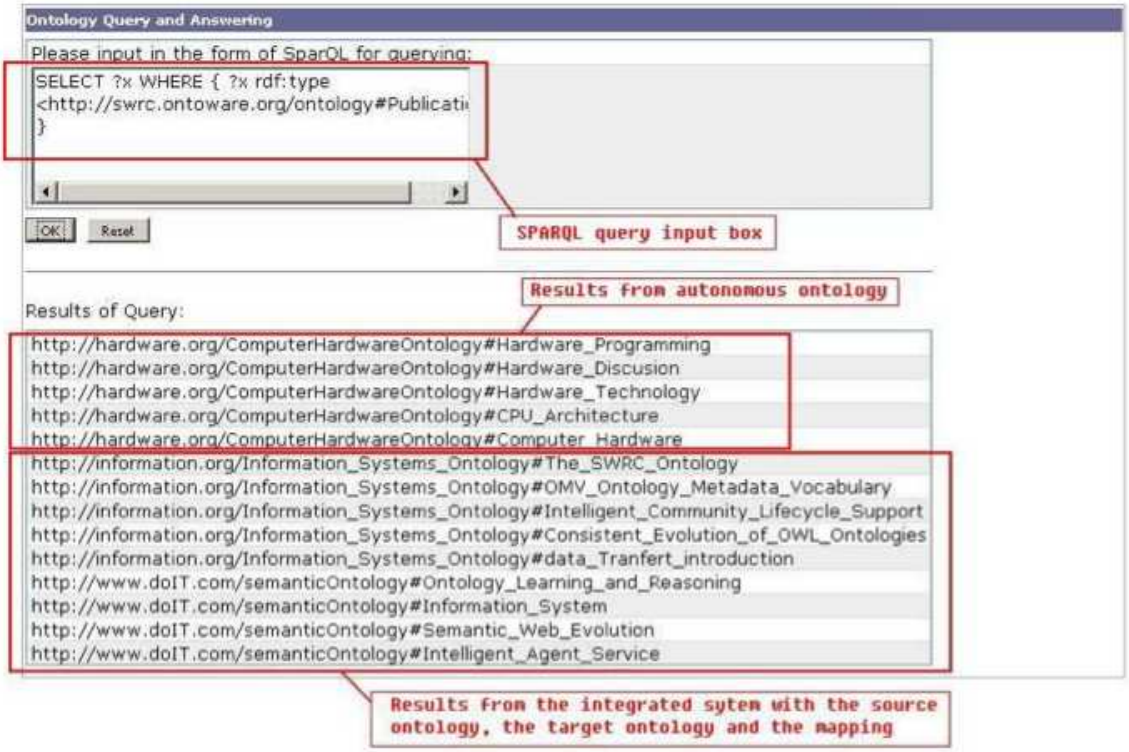

Figure A.8: Query answering use case example.

Bibliography

[ACMHP04] K. Aberer, P. Cudré-Mauroux, M. Hauswirth, and T. Van Pelt. Gridvine: Building internet-scale semantic overlay networks. In *International Semantic Web Conference*, pages 107–121, 2004.

[BS03] Alexander Borgida and Luciano Serafini. Distributed description logics: Assimilating information from peer sources. *J. Data Semantics*, 1:153–184, 2003.

[CDL01] D. Calvanese, G. De Giacomo, and M. Lenzerini. A Framework for Ontology Integration. In *Proceedings of the Semantic Web Working Symposium*, pages 303–316, Stanford, CA, 2001.

[CWW+06] Huajun Chen, Yimin Wang, Heng Wang, Yuxin Mao, Jinmin Tang, Cunyin Zhou, Ainin Yin, and Zhaohui Wu. Towards a semantic web of relational databases: A practical semantic toolkit and an in-use case from traditional chinese medicine. In *International Semantic Web Conference*, pages 750–763, 2006.

[DMS+05] Martin Dzbor, Enrico Motta, Rudi Studer, York Sure, Peter Haase, Asunción Gómez-Pérez, Richard Benjamins, and Walter Waterfeld. Neon - lifecycle support for networked ontologies. In *Proceedings of 2nd European Workshop on the Integration of Knowledge, Semantic and Digital Media Technologies (EWIMT-2005)*, pages 451–452, London, UK, NOV 2005. IEE.

[EHSH05] M. Ehrig, P. Haase, N. Stojanovic, and M. Hefke. Similarity for ontologies - a comprehensive framework. In *13th European Conference on Information Systems*, MAY 2005.

[ES05] Marc Ehrig and York Sure. Foam - framework for ontology alignment and mapping; results of the ontology alignment initiative. In Benjamin Ashpole, Marc Ehrig, Jerome Euzenat, and Heiner Stuckenschmidt, editors, *Proceedings of the Workshop on Integrating Ontologies*, volume 156, pages 72–76. CEUR-WS.org, OCT 2005.

[FKLZ04] E. Franconi, G. M. Kuper, A. Lopatenko, and I. Zaihrayeu. The coDB robust peer-to-peer database system. In *SEBD*, pages 382–393, 2004.

[GPH05] Y. Guo, Z. Pan, and J. Heflin. Lubm: A benchmark for owl knowledge base systems. journal of web semantics. *Journal of Web Semantics*, 3(2):158–182, 2005.

[HBE+04] P. Haase, J. Broekstra, M. Ehrig, M. Menken, P. Mika, M. Plechawski, P. Pyszlak, B. Schnizler, R. Siebes, S. Staab, and C. Tempich. Bibster - a semantics-based bibliographic peer-to-peer system. In *Proceedings of the Third International Semantic Web Conference, Hiroshima, Japan, 2004*, NOV 2004.

[HM05] Peter Haase and Boris Motik. A mapping system for the integration of owl-dl ontologies. In Axel Hahn, Sven Abels, and Liane Haak, editors, *IHIS 05: Proceedings of the first international workshop on Interoperability of heterogeneous information systems*, pages 9–16. ACM Press, NOV 2005.

[HMS04] U. Hustadt, B. Motik, and U. Sattler. Reducing $\mathcal{SHIQ}^-$ Description Logic to Disjunctive Datalog Programs. In *Proceedings of the 9th Conference on Knowledge Representation and Reasoning (KR2004)*, pages 152–162. AAAI Press, June 2004.

[HMS05] U. Hustadt, B. Motik, and U. Sattler. Data Complexity of Reasoning in Very Expressive Description Logics. In *Proceedings IJCAI 2005*, Edinburgh, UK, 2005. Morgan-Kaufmann.

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

[HPS04] Ian Horrocks and Peter Patel-Schneider. Reducing OWL entailment to description logic satisfiability. *J. of Web Semantics*, 1(4):345–357, 2004.

[HRW+06] Peter Haase, Sebastian Rudolph, Yimin Wang, Saartje Brockmans, Raul Palma, Jéróme Euzenat, and Mathieu d'Aquin. D1.1.1 networked ontology model. Technical Report D1.1.1, Universität Karlsruhe, NOV 2006.

[HSH+05] Jens Hartmann, York Sure, Peter Haase, Raul Palma, and Mari del Carmen Suárez-Figueroa. Omv – ontology metadata vocabulary. In Chris Welty, editor, *ISWC 2005 - In Ontology Patterns for the Semantic Web*, NOV 2005.

[HSvH04] P. Haase, R. Siebes, and F. van Harmelen. Peer selection in peer-to-peer networks with semantic topologies. In *Proceedings of the First International IFIP Conference on Semantics of a Networked World: ICSNW 2004, Paris, France, June 17-19, 2004.*, pages 108–125, 2004.

[HW07] Peter Haase and Yimin Wang. A decentralize infrastructure for query answering over distributed ontologies. In *The 22nd Annual ACM Symposium on Applied Computing (SAC'07)*, Seoul, Korea, 2007. To appear.

[Len02] M. Lenzerini. Data integration: a theoretical perspective. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 233–246. ACM Press, 2002.

[MMS+03] A. Maedche, B. Motik, L. Stojanovic, R. Studer, and R. Volz. An infrastructure for searching, reusing and evolving distributed ontologies. In *Proceedings of the Twelfth International World Wide Web Conference (WWW 2003)*, pages 439–448. ACM, 2003.

[MS06] Boris Motik and Ulrike Sattler. A comparison of reasoning techniques for querying large description logic aboxes. In Miki Hermann and Andrei Voronkov, editors, *Proc. of the 13th Int. Conf. on Logic for Programming Artificial Intelligence and Reasoning (LPAR 2006)*, LNCS, Phnom Penh, Cambodia, 2006. Springer. To appear.

[MSS04] B. Motik, U. Sattler, and R. Studer. Query Answering for OWL-DL with Rules. In *International Semantic Web Conference*, pages 549–563, 2004.

[PAG06] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of sparql. In *International Semantic Web Conference*, pages 30–43, 2006.

[PH05a] R. Palma and P. Haase. Oyster - sharing and re-using ontologies in a peer-to-peer community. In *International Semantic Web Conference*, pages 1059–1062, 2005.

[PH05b] Raúl Palma and Peter Haase. Oyster - sharing and re-using ontologies in a peer-to-peer community. In *International Semantic Web Conference - Semantic Web Challenge*, pages 1059–1062, 2005.

[SP04] E. Sirin and B. Parsia. Pellet: An OWL DL reasoner. In *Description Logics*, 2004.

[ST05] L. Serafini and A. Tamilin. Drago: Distributed reasoning architecture for the semantic web. In *Proceedings of the Second European Semantic Web Conference, ESWC 2005, Heraklion, Crete, Greece, May 29 - June 1, 2005*, pages 361–376, 2005.

[TH04] Dmitry Tsarkov and Ian Horrocks. Efficient reasoning with range and domain constraints. In *Description Logics Workshop*, 2004. FaCT++.

[TIM+03] I. Tatarinov, Z. Ives, J. Madhavant, A. Halevy, D. Suciu, N. Dalvi, X. Dong, Y. Kadiyska, G. Miklau, and P. Mork. The piazza peer data management project. *SIGMOD Record*, 32(3), 2003.