DISTRIBUTED ONTOLOGY SYSTEMS FOR OWL ONTOLOGIES WITH LARGE NUMBER OF INSTANCES

by

Xue Ying Chen

A thesis submitted to

the Faculty of Graduate and Postdoctoral Affairs

in partial fulfillment of

the requirements for the degree of

DOCTOR OF PHILOSOPHY

School of Computer Science

at

CARLETON UNIVERSITY

Ottawa, Ontario

January, 2011

Your file Votre référence
ISBN: 978-0-494-81539-7
Our file Notre référence
ISBN: 978-0-494-81539-7

Abstract

Since the inception of ontology systems, we have witnessed both the number of ontologies appearing on the web growing rapidly and their size expanding dramatically. The increase in the number of ontologies brings problems in querying multiple ontologies that are possibly heterogeneous and independently developed, and the expansion of ontology size poses challenges in system efficiency and management. While much attention has been paid to ontology integration systems so as to enable interoperability between different ontologies, the problem of how to handle ontologies of large size has not been sufficiently addressed.

In this thesis, we describe a distributed ontology system (DOS) which is more reliable and scalable than centralized ontology systems where ontologies are stored on a single computer. The idea of DOS is to divide a large ontology into smaller pieces, called ontology fragments, then distribute these fragments over a set of autonomous nodes where each of them is equipped with a reasoner. We formally define DOS by presenting a logical model for DOS and discuss its advantages and challenges in terms of query processing and collaborative ontology development.

In addition, we describe a query processing procedure for conjunctive queries with ontologies that have large number of instance. We prove that our method always returns sound answers for OWL ontologies, and returns complete answers for $\mathcal{SHI}$ ontologies. Furthermore, using the procedure we show how to handle two other reasoning problems, consistency checking introduced by updating ABox and instance realization, in DOS. Finally, we discuss how DOS provides synchronous and non-locking editing for collaborative ontology development.

To my husband

Hong Jin Zhang

Acknowledgements

I would like to thank my thesis supervisor, Professor Michel Dumontier, for his guidance in research, his constant encouragement and patience throughout my thesis.

I would also like to thank all the examiners of my thesis for their time and helpful comments. In particular I would like to thank Professor Amy Felty for her instructive comments on my work.

I am grateful to the School of Computer Science at Carleton University for providing us with a good environment and facilities to complete this research. I would like to thank all the professors who have helped me in broadening my view and knowledge.

Most importantly, I would like to thank my family for their love and support, especially my husband, Hongjin Zhang, thank you for always being there for me.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

The term "ontology" is originally from the field of philosophy that studies the nature of being, existence or reality in general, as well as the basic categories of being and their relations. In computer science and information science, ontology is defined as "a specification of a conceptualization" [42], that is, a formal representation of a set of concepts within a domain and the relationships between those concepts. Ontologies are shared by a community of users and used to reason about the properties of the domain. Nowadays ontologies play an important role in many fields such as the semantic web, knowledge management systems, e-science, bio-informatics and grid applications [105].

In recent years, we have witnessed both the number and the size of ontologies appearing on the web growing rapidly. For example, the yOWL[1][115] ontology contains over 500K instances, NCI thesaurus[2] contains over 80K concepts and increases by 900 new entries each month. Current ontology editors provide either single-user mode or client-server mode (see Figure 1.1 for architecture) for creating, developing and maintaining ontologies [2][33][77][98][111], where the ontology is stored on a single computer.

While centralized ontology storage is convenient in security control and administration, it has a number of disadvantages:

[1]http://ontology.dumontierlab.com/yowl-jbi.owl
[2]http://nciterms.nci.nih.gov/

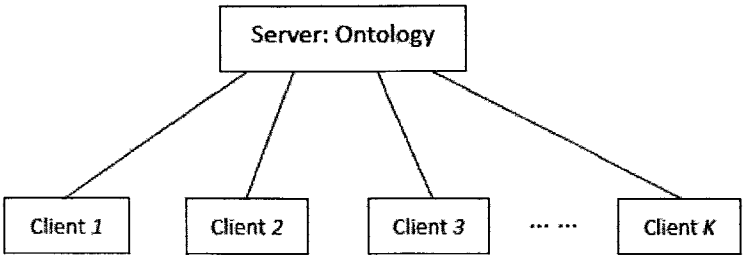

Figure 1.1: Client-Server Architecture

- Single point of failure, which happens when the server's failure brings down the whole system.
- As the number of simultaneous client requests to the server increases, the server can get overloaded and the network can get congested, which will result in slow response to the clients' requests.
- Generally, it is more expensive and difficult to set up the server due to the technical complexity.

Although there exists a number of reasoners based on different reasoning algorithms and optimization techniques, such as RacerPro[3], FaCT++[4] and Pellet[5] based on tableau algorithms, HermiT[6] based on hyper-tableau calculus, and KAON2[7] using disjunctive datalog, they show poor performance in reasoning with large ontologies [17][81][99].

In addition, building and maintaining a large ontology require a team of ontology developers to work collaboratively. The developers are usually geographically distributed and contribute to the ontology in different ways. Tasks for maintaining an ontology include tracking and managing the frequent changes to the ontology,

[3]http://www.racer-systems.com/products/index.phtml
[4]http://owl.man.ac.uk/factplusplus/
[5]http://www.mindswap.org/2003/pellet/index.shtml
[6]http://hermit-reasoner.com/
[7]http://kaon2.semanticweb.org

reconciling conflicting views of the domain from different developers, minimizing the introduction of errors and so on. In a centralized ontology environment, performing these tasks at the same time by different developers may result in an overloaded server or a congested network.

Inspired by distributed database systems [90], we proposed Distributed Ontology Systems (DOS) for managing large ontologies.

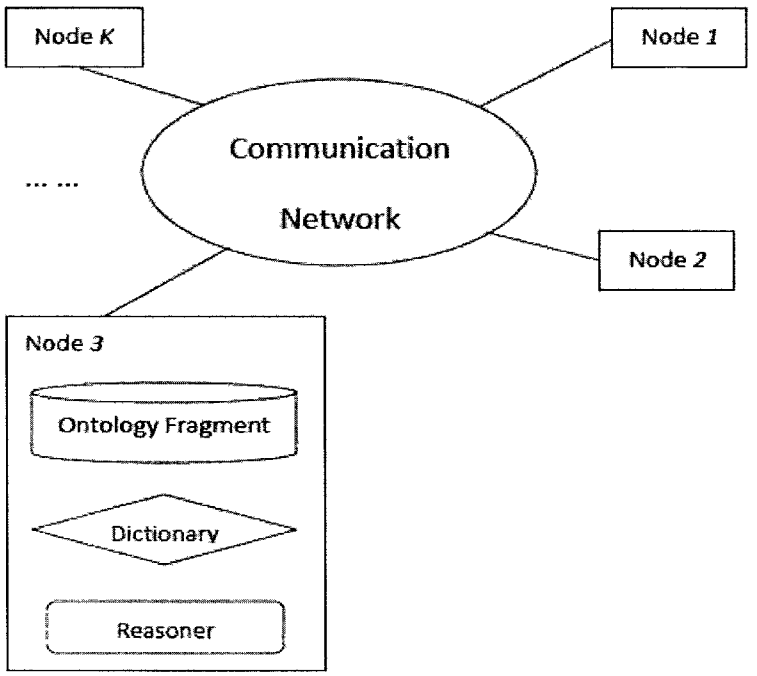

Figure 1.2: Distributed Ontology System Architecture I

In DOS (see Figure 1.2 for architecture), an ontology is partitioned into smaller pieces, called ontology fragments. The fragments are distributed across an arbitrary number of autonomous nodes where each of them is equipped with a reasoner. Users can pose queries through any node, and the queries are viewed as queries posed over the whole ontology instead of the ontology fragment on the node. This architecture has several desirable features:

- Transparency of fragmentation.

How to partition an ontology is the responsibility of the system administrator, that is, users interact with DOS in the same way as they interact with systems with centralized ontologies.

- More flexible replication policy.

  DOS provides more options in the number of copies of ontology fragments and in the locations of the replicas.

- Reliability.

  DOS promises a more reliable system by eliminating single point of failure. In DOS, the failure of a single site or the failure of a communication link is not sufficient to bring down the entire system, therefore, users can still access other parts of the ontology.

- Scalability.

  It is easier and more economical to expand systems in distributed environment. Increasing ontology size can be accommodated by adding processing and storage power to the network. Furthermore, it usually costs much less to put together a system of small computers than to use the equivalent power of a single large machine.

Furthermore, in distributed mode it is possible to improve the system efficiency in query processing if the knowledge base that needs to be accessed becomes smaller, e.g., the queries are evaluated against one or several fragments instead of the whole ontology.

From the above discussion, we can see that DOS has many advantages over systems with centralized ontologies. Although DOS also brings up problems in distributed ontology design and query processing, we are interested in studying some typical ontology reasoning problems and collaborative ontology development in this context.

1.2 Statement of the Problem

In this thesis, we consider three problems: the definition of distributed ontology systems, some typical reasoning problems with ontologies that have large ABoxes in DOS, and collaborative ontology development in DOS.

Firstly, we formalize distributed ontology systems by presenting a logical model for them. A distributed ontology system has three key components: an ontology fragment, a reasoner and a dictionary that contains metadata. Fragmentation (how to divide an ontology) plays an important role in DOS since it determines the content of each fragment. We discuss the cases of fragmentation and summarize the advantages and challenges in terms of query processing and collaborative development for each case.

Secondly, we consider three reasoning problems in DOS: i) conjunctive query answering, ii) consistency checking introduced by updating ABoxes, iii) instance realization. We focus on the case where an ontology has a large ABox, and after fragmentation each ontology fragment contains complete TBox and RBox but incomplete ABox.

The third problem we discuss in this thesis is collaborative ontology development. As preparation for implementing DOS, we discuss the requirements for supporting collaborative development, and describe how to meet those requirements in DOS.

1.3 Contributions

Our contributions include:

- Describe distributed ontology systems to handle large ontologies and present a formal definition for the systems.

- Describe a query processing procedure for conjunctive queries in DOS and prove

that the procedure always returns sound answers for OWL ontologies, and returns complete answers for $\mathcal{SHI}$ ontologies, which correspond to $\mathcal{ALC}$ description logic (a basic description logic providing logical constructors such as $\neg$, $\sqcap$, $\exists r.C$ and $\forall r.C$) extended with transitive roles($\mathcal{S}$), role hierarchy($\mathcal{H}$) and inverse roles($\mathcal{I}$).

- Describe how to handle consistency checking introduced by updating ABoxes and instance realization in DOS by using the procedure for conjunctive queries.
- Describe how to provide synchronous and non-locking collaborative ontology development in DOS.
- "A framework for distributed ontology systems". In proc. of the Second Canadian Semantic Web Working Symposium(CSWWS09).
- "Conjunctive Query Answering in Distributed Ontology Systems for OWL Ontologies with Large ABoxes". In proc. of OWLED09 (short paper).
- "Conjunctive Query Answering in Distributed Ontology Systems for OWL Ontologies with Large ABoxes". submitted to ESWC2011.

1.4 Organization of Thesis

This thesis is organized as follows. Chapter 2 introduces background knowledge and Chapter 3 introduces some distributed systems in databases and AI. Chapter 4 presents a logical model for DOS and discusses the advantages and challenges for each case of fragmentation. In Chapter 5, we discuss three typical ontology reasoning problems in DOS. Chapter 6 shows how to perform collaborative ontology development in DOS. Conclusions and future work are in Chapter 7.

Chapter 2

Background

In this chapter, we introduce ontologies as a knowledge representation and reasoning formalism and discuss the applications of ontologies in the semantic web. We also recall the concepts and terms of basic description logics which are the basis of the web ontology language: OWL.

2.1 Introduction

A knowledge base is a repository for information. The content of a knowledge base depends on the specific domain in which the application is engaged. For example, a knowledge base for publishers may contain information about authors, book titles, bibliography of authors and so on.

Knowledge representation and reasoning is the study of thinking as computational process, that is, how to use what we know in deciding what to do. For the purpose of automated reasoning, researchers have developed symbolic structures for representing knowledge, including classical logics (e.g.,predicate logic) and nonclassical logics (e.g., nonmonotonic logics) [14][56].

Example 1 *Table 2.1 shows an example of knowledge base $\mathcal{O}$ where each axiom or fact is followed by the intuition(in /**/) that it captures. The example is represented in description logic, which is a fragment of predicate logic.*

Axiom (i) and facts (ii), (iii) and (iv) are viewed as explicit knowledge of $\mathcal{O}$ as they are explicitly contained in $\mathcal{O}$. But in addition to that, we can infer more information.

Table 2.1: Example of Knowledge Base

(i). Author $\sqsubseteq$ Human
/* *All authors are human.* */
(ii). Author(Lewis Carroll)
/* *Lewis Carroll is an author.* */
(iii). Book(Alice Adventure in Wonderland)
/* *"Alice Adventure in Wonderland" is a book.* */
(iv). WrittenBy(Alice Adventure in Wonderland,Lewis Carroll)
/* *"Alice Adventure in Wonderland" is written by Lewis Carroll.* */

For instance, from (i) and (ii), we can conclude than "Lewis Carroll" is a human, written as Human(Lewis Carroll), which is viewed as implicit knowledge in $\mathcal{O}$.

A knowledge-based system is able to provide intelligent decisions with justification based on its knowledge base. An early example of knowledge-base system is expert systems which use knowledge bases of human expertise for problem solving or clarifying uncertainties. Expert systems were introduced by researchers from the Stanford University with Dendral system [71] in 1960s and Mycin system [20] in 1970s. Perhaps the most famous knowledge-based system is CYC[1] developed since 1984 by company Cycorp. CYC aims to provide a deep layer of "common sense knowledge" that can be used by other knowledge-intensive programs, and it currently contains more than 177K concepts(axioms) and 1.5 million assertions(facts).

Ontologies are modern examples of knowledge-base systems. Although many ontology definitions can be found in computer science and information science, the common elements in them are representing, reasoning, sharing and reuse of existing domain knowledge [43]. The most common citation in ontology is attributed to T. Gruber, who defined ontology as "a specification of a conceptualization" [42], that is, a formal representation of concepts within a domain and the relationships between those concepts. This definition captures several characteristics of an ontology as a

[1]http://www.cyc.com/

specification of domain knowledge:

- Conceptualization.

  An ontology specifies knowledge in a conceptual way by using symbols to represent concepts and their relationships. Normally, these symbols can be intuitively grasped by humans since their counterparts can be found in our mental model.

- Formality.

  An ontology is expressed in a knowledge representation language that provides a well-defined syntax and semantics. This ensures that the specification of domain knowledge is machine-processable.

- Domain specified.

  The specification in an ontology is limited to knowledge about a particular domain of interest.

- Being shared.

  An ontology reflects an agreement on a domain conceptualization among people in a community.

Ontologies are a core part of the semantic web. The Semantic Web is a set of standards in the representation, publishing, sharing, linking and querying of data and services using web technology. Semantic web is viewed as an evolution from the web where linked document repository is simply provided to a new generation of web where not only the content of the web but also the relationships among the content are understandable and available in a machine processable form. This collection of interrelated datasets in semantic web is also referred as "linked data" since the distributed data across the web is connected by relations defined in ontologies. The semantic web provides an application platform where "information is given well-defined meaning, enabling computers and people to work in cooperation" [73] by adding semantic

annotations to web content and their relationships. The descriptive terms in those annotations are defined in ontologies.

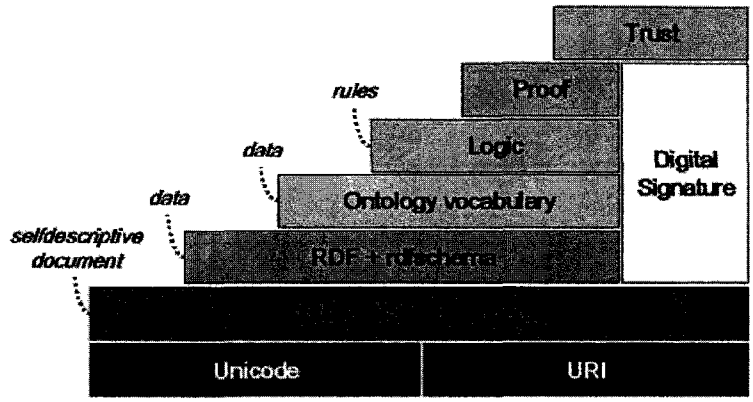

Figure 2.1: Semantic Web Stack

The Semantic Web builds on existing web technology. Figure 2.1 shows the layers of semantic web [4]:

- Unicode is a universal standard encoding system and it allows all human languages to be used (written and read) on the web using one standardized form.

- Uniform Resource Identifier (URI) is a string of a standardized form that allows to uniquely identify resources. The usage of URI is important for a distributed internet system as it provides understandable identification of all resources.

  An international variant to URI is Internationalized Resource Identifier (IRI) which allows usage of Unicode characters in identifier and for which a mapping to URI is defined.

- Extensible Markup Language (XML) with namespace and schema definitions make sure that there is a common syntax used in the semantic web.

  XML is a general purpose markup language for documents containing structured information. An XML document contains elements that can be nested and their

attributes and content. XML namespaces allow to specify different markup vocabularies in one XML document. XML schema serves for expressing schema of a particular set of XML documents.

- Resource Description Framework (RDF) is a graph model for representing information about resources. It is based on triples "subject-predicate-object" to form graph of data. All data in the semantic web uses RDF as the primary representation language. A common syntax for serializing RDF is XML in the RDF/XML form.

  RDF Schema (RDFS) intends to structure RDF resources and provides additional vocabulary for basic reasoning about types and the domain and range of object relations.

  With RDF and RDFSchema (RDFS), it is possible to make statements about objects with URI's or IRI's and define vocabularies that can be referred to. This is the layer where we can give types to resources and links.

- The Ontology layer supports the evolution of vocabularies as it can define relations between the different concepts. An example ontology language is OWL which is a language derived from description logics, and offers more constructs over RDFS. OWL is syntactically embedded into RDF. So like RDFS, OWL provides additional standardized vocabulary.

- The Digital Signature layer provides means for detecting alterations to documents and verification of the origin of the sources.

- The Logic layer provides the feature of first order logic. The user states logical principles and then the computer infers new knowledge by applying these principles to the existing data.

- the Proof layer executes the rules specified in the Logic layer and generates proof for the inference.

- Trust layer is the top layer where agents that want to work with the full-featured semantic web are placed over. It provides trust engines which are constructed based on reasoning engines and digital signatures, therefore, this layer can be developed with rules about which signed assertions are trusted depending on the signers.

In the Ontology layer, the OWL Web Ontology Language [92] is a World Wide Web Consortium[2] (W3C) recommended ontology language. It provides unambiguous syntax, well-defined semantics for representing information as well as a reasoning mechanism with predictable complexity. OWL derives from a decidable fragment of first order logics called description logics.

2.2 Basic Description Logics

Description Logics (DLs) [10] [55] [97] are a family of logic-based knowledge representation formalisms designed to represent and reason about the knowledge of an application domain in a structured and well-understood way. It is a fragment of first-order logic with only unary and binary predicates.

2.2.1 Syntax and Semantics of Description Logics

There are three components of a DL knowledge base: TBox, RBox and ABox. The TBox and RBox contain terminological axioms and role axioms respectively, while the ABox contains assertions. The vocabulary of a DL knowledge base consists of *concepts*, which are unary predicates denoting individuals, and *roles*, which are binary relations denoting relationships between individuals. In addition to atomic concepts

[2]http://www.w3.org/

and roles, all DL systems provide *constructors* to build complex concepts and roles from atomic ones.

We use *interpretation* $\mathcal{I}$ to specify the semantics of DLs. An interpretation $\mathcal{I}$ is a pair $(\Delta^I, \cdot^I)$, where Δ^I is a non-empty set which is the *domain* of the interpretation, and $\cdot^I$ is the interpretation function that assigns to every concept C a set $C^I \subseteq \Delta^I$ and to every role r a binary relation $r^I \subseteq \Delta^I \times \Delta^I$. Given a DL knowledge base $\mathcal{O}$, an interpretation $\mathcal{I}$ is a *model* of $\mathcal{O}$ if $\mathcal{I}$ satisfies all axioms in $\mathcal{O}$.

Table 2.2: $\mathcal{ALC}$ Description Logic

Constructor	**Semantics**
$\top$ (Universal Concept)	Δ^I
$\neg C$ (Negation)	$\Delta^I \setminus C^I$
$C \sqcap D$ (Intersection)	$C^I \cap D^I$
$C \sqcup D$ (Union)	$C^I \cup D^I$
$\exists r.C$ (Full Existential Quantification)	$\{x \in \Delta^I \mid \exists y : (x,y) \in r^I \wedge y \in C^I\}$
$\forall r.C$ (Value Restriction)	$\{x \in \Delta^I \mid \forall y : (x,y) \in r^I \rightarrow y \in C^I\}$
Axioms	**Satisfiability Conditions**
$C \equiv D$	$C^I = D^I$
$C \sqsubseteq D$	$C^I \subseteq D^I$
$a : C$	$a \in C^I$
$r(a,b)$	$(a^I, b^I) \in r^I$

A specific DL is characterized by the constructors it provides. $\mathcal{ALC}$ [104] is a basic description logic. Let C and D be concepts, p and r roles, a and b individuals in the domain, Table 2.2 shows the constructors provided by $\mathcal{ALC}$ and their semantics, axioms appearing in an $\mathcal{ALC}$ knowledge base and their satisfiability conditions.

Table 2.3 lists constructors that can be used to describe more complicated knowledge bases. Further extensions of description logics are done by adding features such as transitive roles($\mathcal{S}$), role hierarchies($\mathcal{H}$), inverse roles($\mathcal{I}$), functional roles($\mathcal{F}$), number restrictions($\mathcal{N}$), qualified number restrictions($\mathcal{Q}$) and nominals($\mathcal{O}$). These

extension can be used in different combinations, for example, OWL DL, which corresponds to $\mathcal{SHOIN}$ [59], is an extension of $\mathcal{ALC}$ with transitive roles, role hierarchies, nominals, inverse roles and number restrictions.

Table 2.3: More Description Logic Constructors

DLs	Semantics
$\mathcal{S}$ (Transitive Role)	$(Trans(r))^I$ $= \{x, y, z \in \Delta^I \mid (x, y) \in r^I \wedge (y, z) \in r^I \rightarrow (x, z) \in r^I\}$
$\mathcal{H}$ (Role Hierarchy)	$(r \sqsubseteq s)^I = r^I \subseteq s^I$
$\mathcal{I}$ (Inverse Role)	$(r^-)^I = \{(x, y) \mid (y, x) \in r^I\}$
$\mathcal{F}$ (Functional Role)	$(Funct(r))^I$ $= \{x, y, z \in \Delta^I \mid (x, y) \in r^I \wedge (x, z) \in \Delta^I \rightarrow y = z\}$
$\mathcal{N}(NumberRestriction)$	$(\geq n\, r)^I = \{x \in \Delta^I \mid \mid \{y \mid (x, y) \in r^I\}\mid \geq n\}$ $(\leq n\, r)^I = \{x \in \Delta^I \mid \mid \{y \mid (x, y) \in r^I\}\mid \leq n\}$
$\mathcal{Q}$ (Qualified Number Restriction)	$(\geq n\, r.C)^I = \{x \in \Delta^I \mid \mid \{y \mid (x, y) \in r^I \wedge y \in C^I\}\mid \geq n\}$ $(\leq n\, r.C)^I \{x \in \Delta^I \mid \mid \{y \mid (x, y) \in r^I \wedge y \in C^I\}\mid \leq n\}$
$\mathcal{O}$ (Nominal)	$\{N_1, N_2, \cdots, N_n\}^I = \{N_1^I, N_2^I, \cdots, N_n^I\}$

The following example shows DL knowledge base about students and the courses that they take.

Example 2 *Table 2.4 is an example of DL knowledge base.*

Table 2.4: Example of DL Knowledge Base

TBox	$FemaleStudent \equiv Female \sqcap Student$	
	$MaleStudent \equiv Male \sqcap Student$	
	$FemaleStudent \sqcap MaleStudent \sqsubseteq \bot$	
	$\forall Attends.Course \sqsubseteq Student$	
RBox	$RegisterFor \sqsubseteq Attends$	
ABox	$Student(Ann)$	$Female(Ann)$
	$Student(Bob)$	$Male(Bob)$
	$Student(Carol)$	$Student(Daisy)$
	$Course(SCS1001)$	$Course(SCS3005)$
	$Attends(Ann, SCS1001)$	$RegisterFor(Carol, SCS1001)$
	$Attends(Bob, SCS3005)$	$RegisterFor(Daisy, SCS3005)$

In TBox, the first two axioms define concepts FemaleStudent/MaleStudent as

students who are female/male. The third axiom means that nobody can be both female student and male student at the same time. The fourth axiom says that only students can attend courses.

The axiom in RBox says that a student that registers for a course also attends the course.

The ABox contains the set of facts about individuals, including whether they are students, their gender, courses they attend or register for. Here, Ann, Bob, Carol and Daisy are students. Ann is female, and Bob is male. SCS1001 and SCS3005 are two Courses. Ann and Bob attend courses SCS1001 and SCS3005, respectively, Carol and Daisy register for courses SCS1001 and SCS3005, respectively.

2.2.2 Description Logic Inference

A description logic knowledge base is able to perform specific kinds of reasoning. Given a TBox $\mathcal{T}$, reasoning problems for concepts include:

- Satisfiability: A concept C is *satisfiable* with respect to $\mathcal{T}$ if and only if there exists a model $\mathcal{I}$ of $\mathcal{T}$ such that C^I is nonempty.
- Subsumption: A concept C is *subsumed* by a concept D with respect to $\mathcal{T}$ if and only if $C^I \subseteq D^I$ for every model $\mathcal{I}$ of $\mathcal{T}$.
- Equivalence: Two concepts C and D are *equivalent* with respect to $\mathcal{T}$ if and only if $C^I = D^I$ for every model $\mathcal{I}$ of $\mathcal{T}$.
- Disjointness: Two concepts C and D are *disjoint* with respect to $\mathcal{T}$ if and only if $C^I \cap D^I = \emptyset$ for every model $\mathcal{I}$ of $\mathcal{T}$.

Note that subsumption, equivalence and disjointness of concepts can be reduced to satisfiability in a description logic that is equivalently or more expressive than $\mathcal{ALC}$(see Section 2.2.4 in [10]).

Given an ABox $\mathcal{A}$ and a TBox $\mathcal{T}$, reasoning tasks associated with *assertions* are:

- Consistency: $\mathcal{A}$ is consistent with respect to $\mathcal{T}$ if and only if there exists a model $\mathcal{I}$ for both $\mathcal{A}$ and $\mathcal{T}$.

- Instantiation: a is an instance of concept C with respect to $\mathcal{A}$ if and only if $a^I \in C^I$ for all models of $\mathcal{A}$.

- Conjunctive query answering:

$$A(\bar{x}) \leftarrow B_1(\bar{y_1}), \cdots, B_n(\bar{y_n}) \tag{2.1}$$

  where $A(\bar{x})$ and $B_i(\bar{y_i})(i = 1, \cdots, n)$ are concepts or roles in a DL knowledge base, A and B_i are unary or binary predicates, $\bar{x}$ and $\bar{y_i}$ are lists of variables and constants that match the arity, i.e., number of attributes of x, b_i respectively. $\bar{x}$ captures a set of tuples t such that t^I satisfies $B_1(\bar{y_1}), \cdots, B_n(\bar{y_n})$ for all models of $\mathcal{A}$ and $\mathcal{T}$.

2.2.3 DL Knowledge Bases vs. Databases

The main difference between databases and knowledge bases is that while databases concentrate on managing and manipulating a large set of relatively simple data, knowledge bases provide support for inference which involves fewer but more complex data, that is, finding answers about a knowledge base which had not been explicitly described in the knowledge base. We compare databases with DL knowledge bases in the following two aspects:

1. Closed- vs. open-world semantics

   The closed-world semantics can be understood as a computational reinterpretation of *negation* that is known as "*weak negation*" or "*negation as failure*". It assumes that what cannot be proven is false. For example, in a knowledge base

about flights, if we cannot find information about direct flights from Ottawa to Shanghai, then with closed-world semantics, we can conclude that there are no direct flights from Ottawa to Shanghai. However, with open-world semantics, we cannot reach the same conclusion unless we find information in the knowledge base that explicitly states that there are no direct flights from Ottawa to Shanghai.

This means that closed-world semantics assumes the information in a knowledge base is complete, while open-world semantics assumes that it is incomplete. Closed-world semantics are applied to database systems such as relational databases, to logic programs, and to some non-classical logic reasoning like commonsense reasoning. Open-world semantics is applied to classical logic reasoning including reasoning in DL knowledge bases.

2. Monotonic vs. non-monotonic reasoning

   Knowledge representation and reasoning formalisms can be classified into monotonic or non-monotonic according to whether expanding a knowledge base invalidates conclusions drawn from the original knowledge base or not. If expanding a knowledge base does not invalidate those conclusions, then the representation and reasoning formalism is monotonic, otherwise, it is non-monotonic. For instance, commonsense reasoning is non-monotonic. Consider in a knowledge base about elephants, we do not find information about "Elephants are pink", with commonsense reasoning, we conclude that "Elephants are not pink". However, this conclusion will not be true if the knowledge base incorporates a statement "There is a pink elephant in India".

   Knowledge representation and reasoning formalisms operating under the closed-world semantics are usually non-monotonic. Description logics are examples of monotonic reasoning.

2.3 Web Ontology Languages

Since the inception of the semantic web, the development of languages for modelling ontologies has been a key task. In this section, we introduce web ontology languages: RDF+RDFS, OWL 1 and OWL 2.

2.3.1 RDF and RDF Schema

An early initiative to standardize ontology languages by W3C resulted in Resource Description Framework (RDF) [64] and RDF Schema (RDFS) [13]. RDF is a graph based model for representing information about resources in the World Wide Web. It can be used to represent metadata about Web resources, such as the title, author and creation date of a Web page.

An RDF statement has three parts: *subject* which identifies the thing that the statement is about, *predicate* which identifies the property of the subject, and *object* which identifies the value of that property. An RDF statement can be represented using RDF graphs where the subject and object are nodes and the predicate is an arc.

Example 3 *Consider the Web page "http://www.college.ca" has a creator whose name is Adam, and the Web page's language is English.*

Here in the first statement, the subject is "http://www.college.ca", the predicate is "creator", and the object is "Adam". In the second statement, the subject is "http://www.college.ca", the predicate is "language", and the object is "English". The statements can be represented by the graph shown in Figure 2.2.

There are two kinds of syntax for recording and exchanging RDF graphs. One is XML-based syntax, called RDF/XML, which is widely accepted because it was introduced among the other W3C specifications defining RDF. The other is Notation

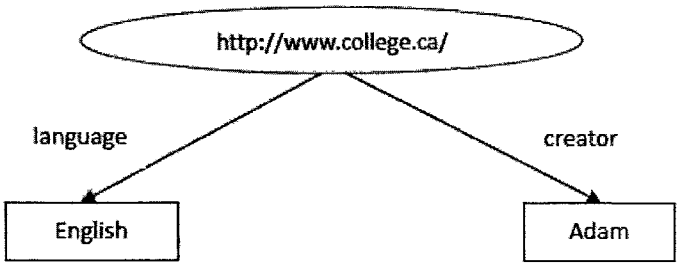

Figure 2.2: Example RDF Graph

3 (or N3) [7] which has become popular in recent years due to its compactness and readability.

Example 4 *Table 2.5 and 2.6 show the RDF graph in Example 3 serialized in XML and Notation 3 syntax, respectively.*

Table 2.5: Example of RDF/XML

```
<?xml version="1.0">
<rdf: RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

        <rdf:Description rdf:about="http://www.carleton.ca">
                <creator>Adam Carleton<creator>
                <language>English<language>
        </rdf:Description>

 </rdf:RDF>
```

Table 2.6: Example of RDF in Notation 3

```
<http://www.carleton.ca>
        creator "Adam Carleton";
        language "English".
```

In general, RDF uses URI reference as the basis of its mechanism for identifying the subjects,predicates, and objects in statements. A URI reference (or URIref) is a URI, together with an optional fragment identifier at the end. For example,

the URI reference "*http://www.carleton.ca/index.html#section2*" consists of the URI "*http://www.carleton.ca/index.html*" and (separated by the # character) the fragment identifier "section2". A set of URIrefs, particularly a set intended for a specific purpose, is called a vocabulary.

RDF Schema, RDF's vocabulary description language, is a semantic extension of RDF. It provides classes and properties for describing groups of related resources and the relationships between these resources, see [13] for lists of RDFS classes and properties. RDF Schema vocabulary descriptions are written in RDF. The following example specifies the hierarchy of human beings in RDFS.

Example 5 *Table 2.7 specifies that Man and Woman are subclasses of Human, Father is a subclass of Man, and Mother is a subclass of Woman.*

2.3.2 OWL 1

RDF Schema provides basic elements for describing ontologies, but it was soon found to be limited in expressive power [59]. Therefore, an expressive web ontology language - OWL - became a W3C recommendation in February 2004.

An important issue for the design of OWL was the trade-off between expressivity of the language and scalability of reasoning. For this purpose, OWL 1 comes as a family of three language variants of increasing expressive power: OWL Lite, OWL DL and OWL Full [92].

- *OWL Lite* corresponds to $\mathcal{SHIF}$. It supports classification hierarchy and simple constraints.

- *OWL DL* corresponds to $\mathcal{SHOIN}$. It provides users with the maximum expressiveness while retaining computational completeness (all conclusions are guaranteed to be computable) and decidability (all computations will finish in finite time).

Table 2.7: Example of RDF Schema

```
<?xml version="1.0">
<rdf: RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
          xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
          xml:base="http://www.carleton.ca/examples/animals">

<rdf:Description rdf:ID="Human">
      <rdf:type rdf:resource"http://www.w3.org/2000/01/rdf-schema#Class"/>
</rdf:Description>

<rdf:Description rdf:ID="Man">
      <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
      <rdfs:subClassOf rdf:resource="#Human"/>
</rdf:Description>

<rdf:Description rdf:ID="Woman">
      <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
      <rdfs:subClassOf rdf:resource="#Human"/>
</rdf:Description>

<rdf:Description rdf:ID="Father">
      <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
      <rdfs:subClassOf rdf:resource="#Man"/>
</rdf:Description>

<rdf:Description rdf:ID="Mother">
       <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
     <rdfs:subClassOf rdf:resource="#Woman"/>
</rdf:Description>

 </rdf:RDF>
```

- *OWL Full* provides users with the maximum expressiveness and the syntactic freedom of RDF with no computational guarantees. For example, in OWL Full a class can be treated simultaneously as a collection of individuals and as an individual in its own right.

2.3.3 OWL 2

Practical experience with OWL 1 has revealed that OWL 1 lacks some constructs that are often useful for modelling complex domains [95][96]. As a response, a Working Group of W3C developed an even more expressive ontology language based on OWL 1 and decided to call it OWL 2 in April 2008.

OWL 2 [60][29] corresponds to the description logic $\mathcal{SROIQ}(D)$, which is obtained by extending $\mathcal{SROIQ}$ with datatypes restriction construct [55]. Description logic $\mathcal{SROIQ}$ is an extension of $\mathcal{SHOIQ}$ with the following constructors:

1. Negated role assertions, e.g., $\neg Love(Alice, Bob)$.

2. Complex role inclusion axioms of the form $R \circ S \sqsubseteq R$ and $S \circ R \sqsubseteq R$ where R and S are roles. For example, given an axiom $Owns \circ HasPart \sqsubseteq Owns$, consider the fact that each computer contains a video card $Computer \sqsubseteq \exists HasPart.VideoCard$, we can conclude that an owner of a computer is also an owner of a video card $\exists Owns.Computer \sqsubseteq \exists Owns.VideoCard$.

3. Disjoint roles, e.g., the roles *Siblings* and *ParentOf* are disjoint.

4. Reflexive, irreflexive and antisymmetric roles, e.g., the roles *Knows*, *ProperPartOf* and *ParentOf* are, respectively, reflexive, irreflexive and antisymmetric.

5. Universal role $\mathcal{U}$, which is a prominent feature of hybrid logics [21].

6. Concepts of the form $\exists R \cdot Self$ which is used to express "local reflexivity" of a role, e.g., $Narcist \equiv \exists Likes \cdot Self$.

OWL 2 specification also identifies three profiles(sub-languages or syntactic subsets): OWL 2 EL, OWL 2 QL and OWL 2 RL, each of which provides different expressive power and targets different application scenarios.

- OWL 2 EL is based on the $\mathcal{EL}++$ which is a lightweight description logic that promises sound and complete reasoning in polynomial time. The main reasoning problem in this profile is classification which computes the subsumption relation between all named classes in an ontology. Reasoning in this profile can be implemented in polynomial time in the size of the ontology [6].

- OWL 2 QL is based on the DL-Lite family of description logics [27]. The main reasoning problem in this profile is conjunctive query answering which can be implemented in LogSpace using standard relational database techniques.

- OWL 2 RL is designed to enable the implementation of polynomial time reasoning algorithms using rule-extended database techniques operating directly on RDF triples [47].

Chapter 3

Distributed Systems

Since the inception of computer network, distributed processing has become a popular topic in almost every area of computer science. In this chapter, we introduce some distribute systems in database area and Artificial Intelligence (AI), including data integration, relational distributed databases and multi-agent systems.

3.1 Introduction

The term "distributed computing" (or "distributed processing") now is one of the most abused term in computer science. It has been used to refer to a wide range of systems such as multiprocessor systems, distributed data processing, multi-agent systems, computer networks and so on.

In [90], a "distributed computing system" is defined as a number of autonomous processing elements that are interconnected by a computer network and cooperate in performing their assigned tasks. The "processing element" refers to a computing device that can execute a program on its own, and different processing elements are not necessarily homogeneous. There are three elements can be distributed: function, data and control. In a distributed system, various functions of a computer system could be assigned to various pieces of hardware and software. Data used by a number of applications may also be distributed to a number of processing sites. The control of the execution of various tasks might be distributed instead of being performed by one computer system.

There exists a number of criteria for classifying distributed computing systems.

Some of them are as follows [3]:

- Degree of coupling, which refers to how closely the processing elements are connected together.

  This can be measured as the ratio of the amount of data exchanged to the amount of local processing performed in executing a task. If the communication is done over a computer network, there exists weak coupling among the processing elements. If computer components such as primary memory or secondary storage devices are shared, then there exists strong coupling.

- Interconnection structure, which refers to the underlying network topology such as point-to-point, star interconnection and so on.

- Independence of components. The processing elements might depend on each other quite strongly when executing tasks, or this interdependence might be as weak as passing messages at the beginning of execution and reporting results at the end.

- Synchronization. Processing elements might be maintained by a synchronous or asynchronous means.

The main reason for the popularity of distributed computing systems is that distributed processing better corresponds to the organizational structure of today's widely distributed enterprises. Consequently, many of current applications of computer technology are inherently distributed, such as E-Commerce over the Internet and multimedia applications. Furthermore, there exists a fundamental advantage behind distributed processing which is that distributed processing corresponds to the idea of solving big and complicated problems by using the divide-and-conquer rule. That is, if the necessary software support for distributed processing can be developed, it might be possible to solve a complicated problem by dividing it into smaller pieces

and assigning them to different software groups which work on different computers but toward the execution of a common task.

3.2 Data Integration Systems

Data integration studies the problem of providing unified access to a set of possibly heterogeneous, autonomous data sources [15][52][54][72]. In data integration systems (see figure 3.1 for architecture), there exists a global schema and a set of data sources where each of them has a local schema describing their own data. Data sources are autonomous and able to process queries posed over their local data, while queries expressed in global schema will be processed upon all the data sources. Therefore, modelling the relationship between the global schema and the data sources is a crucial aspect of data integration.

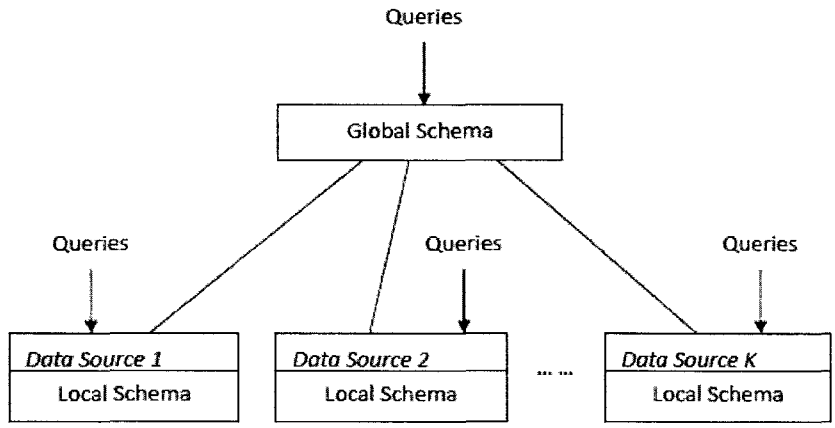

Figure 3.1: Architecture of Data Integration

There are three approaches that have been proposed for modelling the relationship between the global schema and the data sources.

- Global-as-view (GAV), which requires that the global schema is expressed in terms of the data sources, e.g., MOMIS [5] and IBIS [24].

- Local-as-view (LAV), which requires the global schema to be specified independently from the sources, and the relationships between the global schema and the sources are established by defining each source as a view over the global schema, e.g., information manifold [67] and the system described in [75].

- Peer-to-peer, also called GLAV, requires the relationships between the global schema and the sources are established by making use of both LAV and GAV assertions [41][28].

A basic service of data integration systems is to provide users a reconciled view of the underlying data sources, that is, to answer queries posed in terms of the global schema. For this purpose, a data integration system should be able to reformulate a query expressed in global schema into a set of queries expressed over the local schemas using the mappings defining the relationships between the global schema and the local schemas. The following is an example of data integration using global as view.

Example 6 *Consider two relational data sources $DB1$ and $DB2$ containing data about students and the courses that they registered:*

$$\begin{aligned} DB1: &\quad Enrolled(StudentNo, SName, Dept) \\ DB2: &\quad Registered(StudentNo, CourseID, Year) \\ &\quad Course(CourseID, CName) \end{aligned}$$

A global schema V is defined as a view over data in $DB1$ and $DB2$,

$$\begin{aligned} V(SName, CName) \leftarrow\ & Enrolled(StudentNo, SName, Dept), \\ & Registered(StudentNo, CourseID, Year), \\ & Course(CourseID, CName) \end{aligned}$$

To process query q over the global schema,

$$q(SName, CName) \leftarrow V(SName, CName)$$

q is reformulated into two queries q_1 and q_2 over the local schemas,

$$\begin{aligned} q_1(StudentNo, SName) &\leftarrow Enrolled(StudentNo, SName, Dept) \\ q_2(StudentNo, CName) &\leftarrow Registered(StudentNo, CourseID, Year), \\ &\quad Course(CourseID, CName) \end{aligned}$$

Then q_1 and q_2 are used to process query q,

$$q(SName, CName) \leftarrow q_1(StudentNo, SName), q_2(StudentNo, CName)$$

3.3 Relational Distributed Database Management Systems

Distributed database management systems aim to provide a more reliable and efficient management for large databases [12][90]. In a distributed database system (see figure 3.2 for architecture), a database is divided into smaller pieces, called database fragments, which are distributed across the network. It is possible that fragments are overlapped. A database fragment is a subset of the whole database, and the union of fragments equals the database. Therefore, local schemas are parts of the global schema. All the queries are expressed in global schema and evaluated against the whole database instead of the database fragments.

Since each site in the network contains only a subset of the database, information about the distribution of fragments is important for query processing. There are two fundamental fragmentation strategies: horizontal and vertical. In addition, there is possibility of nesting fragments in a hybrid fashion [88][108].

- *Horizontal fragmentation* partitions a relation(table) along its tuples(rows in tables).
- *Vertical fragmentation* partitions a relation into a set of smaller relations.
- *Hybrid fragmentation* performs both horizontal and vertical partition on a relation.

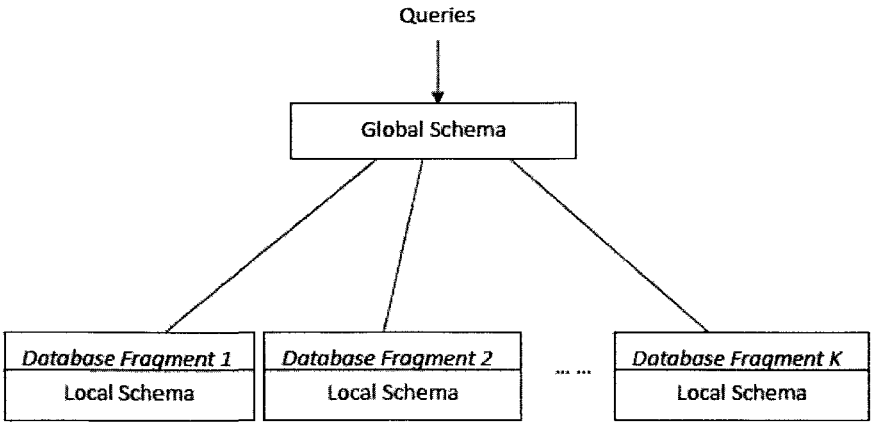

Figure 3.2: Architecture of Distributed Database Systems

The following is an example of horizontal fragmentation.

Example 7 *Given a relation as shown in Table 3.1,*

Table 3.1: Example of Database Fragmentation

Project	PNo	PName	Manager	Budget	Location	Duration
	001	Design	Adam	10k	Ottawa	6 months
	002	Development	Bob	30k	Toronto	18 months
	003	Test	Cathy	10k	Ottawa	3 months
	004	Documentation	David	5k	Toronto	3 months
	005	Installation	Emma	5k	Ottawa	3 months
	006	Maintenance	Fred	10k	Ottawa	12 months

A horizontal fragmentation partitions the above table into two fragments F_1 *and* F_2 *as shown in Table 3.2 and 3.3, respectively.*

Table 3.2: Example of Database Fragment F_1

Project	PNo	PName	Manager	Budget	Location	Duration
	001	Design	Adam	10k	Ottawa	6 months
	002	Development	Bob	30k	Toronto	18 months
	003	Test	Cathy	10k	Ottawa	3 months

Table 3.3: Example of Database Fragment F_2

Project	PNo	PName	Manager	Budget	Location	Duration
	004	Documentation	David	5k	Toronto	3 months
	005	Installation	Emma	5k	Ottawa	3 months
	006	Maintenance	Fred	10k	Ottawa	12 months

These two fragments can be described using the following statements:

$$F_1(PNo, PName, Manager, Budget, Location, Duration) \leftarrow$$
$$Project(PNo, PName, Manager, Budget, Location, Duration), PNo \leqslant 003$$
$$F_2(PNo, PName, Manager, Budget, Location, Duration) \leftarrow$$
$$Project(PNo, PName, Manager, Budget, Location, Duration), PNo \geqslant 004$$

When a query is posed over a distributed database system, it will be decomposed into a set of sub-queries expressed in local schemas, then the sub-queries are optimized using the metadata about the distribution of fragments, finally, the sub-queries are passed onto the sites where the corresponding fragments locate.

Example 8 *Example 7 continued. Consider a query posed over the table Project:*

$$q(PNo, Budget, Duration) \leftarrow$$
$$Project(PNo, PName, Manager, Budget, Location, Duration), PNo \leqslant 003$$

Query decomposition produces the following sub-queries,

$$q(PNo, Budget, Duration) \leftarrow$$
$$F_1(PNo, PName, Manager, Budget, Location, Duration), PNo \leqslant 003$$
$$q(PNo, Budget, Duration) \leftarrow$$
$$F_2(PNo, PName, Manager, Budget, Location. Duration), PNo \leqslant 003$$

then the sub-queries are optimized using the statements describing F_1 and F_2, the result is below:

$$q(PNo, Budget, Duration) \leftarrow$$
$$F_1(PNo, PName, Manager, Budget, Location, Duration)$$

Distributed database management systems provide more efficient and more reliable services than the systems where databases are centralized on a single computer. In distributed database systems, a query over a large database can be decomposed into a set of sub-queries which are evaluated against database fragments that are usually smaller than the original database. Furthermore, for users that frequently access a certain part of database, we can place a fragment containing the corresponding data on their local machine. Distributed database systems are more reliable with the use of replica of fragments, for instance, a failed site can be replaced by a backup site with a replica fragment.

3.4 Multi-agent Systems

Intelligent agents have been a traditional research topic in AI. An intelligent agent is a computer system that is situated in some environment and that is capable of autonomous actions in this environment in order to meet its delegated objectives. Figure 3.3 shows an agent in its environment, where the agent takes sensory input from the environment, and produces as output actions that affect its environment.

[93] classifies four properties of environment:

- Accessible vs. inaccessible.

  An accessible environment is the one in which the agent can obtain complete, accurate, up-to-date information about the environment's state. The more accessible an environment is, the simpler it is to build agents to operate in it.

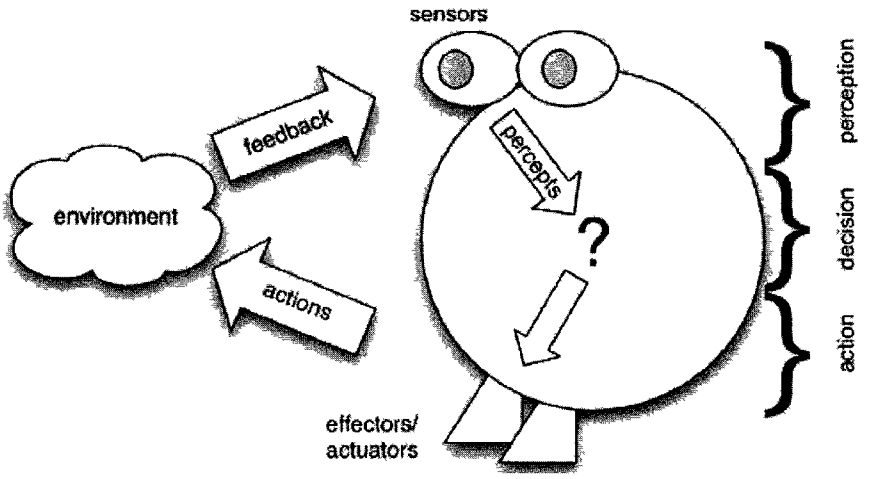

Figure 3.3: An Agent in Environment (Source [116])

- Deterministic vs. non-deterministic.

  A deterministic environment is one in which any action has a single guaranteed effect. That is, there is no uncertainty about the state that will result from performing an action. Non-deterministic environment present greater problems for the agent design.

- Static vs. dynamic.

  A static environment is one that can be assumed to remain unchanged except by the performance of action by the agent. On the contrary, a dynamic environment is one that has other processes operating on it, and which hence changes in ways beyond the agent's control.

- Discrete vs. continuous.

  An environment is discrete if there are fixed finite number of actions and percepts in it.

In general, the most complex kind of environment is on that is inaccessible, non-deterministic, dynamic and continuous.

The study of multi-agent systems focuses on systems in which many intelligent agents interact with each other and work together to achieve a common goal. Using a shared ontology and a shared set of communication conventions, agents can cooperate by means such as voting [11], bargaining [32] and arguing [16]. [117] suggests three kinds of behavior of agents in multi-agent systems: reactivity, proactiveness and social ability.

- Reactivity refers that intelligent agents are able to perceive their environment and respond in a timely fashion to changes that occur in the environment.
- Proactiveness means that agents are capable of exhibited goal-directed behavior by taking the initiative in order to satisfy their design objectives.
- Social ability means that agents are capable of interacting with other agents.

There are a number of languages developed specifically for agent communication, including Knowledge Query and Manipulation Language (KQML) and Knowledge Interchange Format (KIF) in early 1990's [79]. Now the most popular one is Agent Communication Language developed by Foundation for Intelligent Physical Agents, also known as FIPA ACL [39]. FIPA ACL is supported by Java Development Environment (JADE) which intends to provide software package and tools to allow Java developers to deploy and redeploy FIPA agent systems [8].

Chapter 4

A Framework for Distributed Ontology Systems

In this chapter, we describe a logical model for distributed ontology systems and discuss cases of ontology fragmentation.

4.1 Introduction

Semantic technologies give rise to new opportunities in many scientific disciplines where complex knowledge needs to be modelled, e.g., many OWL ontologies have been created and extensively used in the clinical sciences such as GALEN[1] describing around 50K concepts and SNOMED CT[2] containing over 350K concepts. The current centralized management of large ontologies, where an ontology is stored on a single computer, is confronting challenges such as single point of failure, overloaded server and network and so on. Therefore, we propose the distributed ontology system(DOS) for managing large ontologies.

The key idea of a DOS is to partition a large ontology into smaller pieces, called ontology fragments, and distribute these fragments across a number of autonomous nodes where each of them is equipped with a reasoner. A query can be posed over any node, but the query is viewed as being posed over the whole ontology instead of certain ontology fragments. A master node is the node that accepts the query from the user. Master node is responsible for coordinating the other nodes when performing tasks such as evaluating the query and returning the results to the user,

[1]http://www.opengalen.org
[2]http://www.snomed.org

synchronization between different copies of a fragment and so on.

DOS has some advantages over centralized ontology systems. For example, DOS is more reliable since the failure of single site now is not capable of bringing down the whole system, and overloaded server and network in centralized ontology systems can be avoided in DOS by distributing replicated data across the network. DOS also poses some challenges such as query processing in a distributed environment and synchronization between ontology replicas and so on.

In the rest of this chapter, we first introduce some related work, then present a logical model for DOS and discuss the cases of ontology fragmentation.

4.2 Related Work

We discuss here two threads of research work: ontology integration and ontology modularization.

4.2.1 Ontology Integration

The main task of an ontology integration system (OIS) [26] [85] [91] is to facilitate knowledge exchange between ontologies that are possibly heterogeneous and independently developed. Ontologies that are independently developed may differ in many aspects such as using different ontology languages, or using different terms to describe the same concept. For example, in ontology $\mathcal{O}_1$, "people who write novels" is defined as "writer", in ontology $\mathcal{O}_2$, "people who write novels" is defined as "author", to integrate these two ontologies, we need to express the relations between them, that is, "writer" and "author" are two equivalent concepts. In order to enable knowledge exchange between different ontologies (local ontology), OISs semantically combine the information in them, and provide the user with a unified view (global ontology) through which the user can query those combined ontologies.

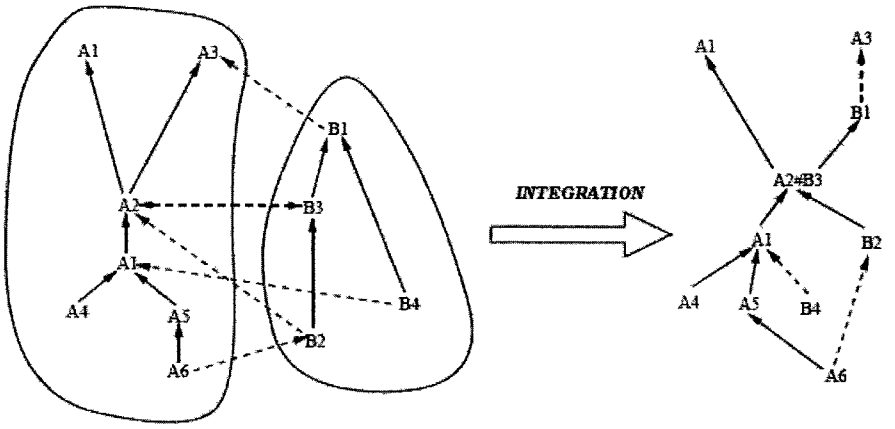

Figure 4.1: Integrating Two Ontologies (Source: [78])

Figure 4.1 shows integrating two ontologies with some relationships between them: arrows are subclasses and their parents, double-headed arrows are synonyms, and dashed lines are relationships between the two ontologies. On the right of the figure is the integrated ontology.

[85] summarizes that OISs deal with three problems:

- Mapping discovery.

  Given two ontologies, OISs should be able to find the similarities between them, i.e., determine which concepts and roles represent similar notions, and so on. [68] [94] discuss finding mappings (semi-) automatically in both databases and ontologies.

- Declarative representations of mappings.

  After finding mappings between two ontologies, OISs should represent them to enable reasoning with mappings, e.g., OntoMerge system[36] and MAFRA framework [80] express the correspondence between two ontologies as a set of bridging axioms relating classes, properties and instances of the two source ontologies.

- Performing reasoning tasks with defined mappings.

Similar to database integration, we can classify OISs into three categories according to how mappings are expressed:

- Global-as-view: Mappings associate a view expressed using concepts or roles in local ontologies to each concept or role in the global ontology, e.g., DOLCE [44] and SUMO [89].

- Local-as-view: Mappings associate each concept or role in a local ontology using concepts or roles in the global ontology.

- Peer-to-peer: Mappings are defined between local ontologies using each other's concepts or roles, e.g., Drago [37] and KAONP2P [63].

Sometimes, peer-to-peer ontology integration systems [30] [38][58] [63] [107] are also called distributed ontology systems, however, they are not the same as DOS defined in this thesis. In peer-to-peer OISs, there is no central control, all nodes throughout the network are autonomous. While in DOSs, there exists a master node coordinating other nodes' behavior though the master node is not fixed. In addition, OISs focus on solving the similarity and heterogeneity between ontologies, but DOSs concentrate on how to manage a large ontology in order to provide a reliable and efficient system to the user. However, if the peer-to-peer OIS completely fills up the semantic gap between local ontologies, it can be viewed as a DOS defined in this thesis.

4.2.2 Ontology Modularization

An ontology is a specification of a conceptualization. It actually provides an agreement on a set of concepts for a domain of interest and describes the relationships between the concepts. When developing an ontology, it is helpful if the engineer can

reuse information extracted from external, existing ontologies. For example, an ontology engineer is building a flower ontology $\mathcal{P}$ which specifies the kinds of flowers and the places they grow. The engineer working on $\mathcal{P}$ wants to include the climate of some specific places such as "Rocky Mountains" and "Tibet" in his ontology by importing the information that he needs from a geography ontology $\mathcal{O}$, ideally, only a subset of $\mathcal{O}$ that contains information about Rocky Mountains and Tibet.

Ontology modularization, which is motivated by partial reuse of ontology, has drawn attention from many researchers in ontology engineering [31][45][46][51][74][106]. A logic-based definition for modules with respect to a given set of concept or role names in an ontology is described in [46][51][66]. That is, given an ontology $\mathcal{O}$ and a signature Σ which specifies a set of concept or role names, we say that $\mathcal{M}$ is a module of $\mathcal{O}$ with respect to Σ if all consequences of $\mathcal{O}$ that can be expressed over Σ are also consequences of $\mathcal{M}$. $\mathcal{O}$ is said to be a deductive conservative extension of $\mathcal{M}$.

Formally,

Definition 1 *([46]) Let $\mathcal{O}$ and $\mathcal{M} \subseteq \mathcal{O}$ be two $\mathcal{L}$-ontologies, and $\mathcal{S}$ a signature over $\mathcal{L}$. We say that $\mathcal{O}$ is a deductive $\mathcal{S}$-conservative extension of $\mathcal{M}$ with respect to $\mathcal{L}$, if for every axiom α over $\mathcal{L}$ with $Sig(\alpha) \subseteq \mathcal{S}$, we have $\mathcal{O} \models \alpha$ if and only if $\mathcal{M} \models \alpha$. We say that $\mathcal{O}$ is a deductive conservative extension of $\mathcal{M}$ with respect to $\mathcal{L}$ if $\mathcal{O}$ is a deductive $\mathcal{S}$-conservative extension of $\mathcal{M}$ with respect to $\mathcal{L}$ for $\mathcal{S} = Sig(\mathcal{M})$.*

The intuition behind the above definition is that $\mathcal{M}$ captures the meaning of $\mathcal{S}$ in $\mathcal{O}$, therefore, to reuse concepts or roles from $\mathcal{S}$, we can import $\mathcal{M}$ instead of the whole ontology $\mathcal{O}$. However, deciding conservative extensions is computationally expensive. For $DL\text{-}Lite_{horn}$[3], complexity grows from polynomial to coNP-complete, and for $DL\text{-}Lite_{bool}$[4], complexity grows from NP-complete to Π_2^p-complete [70]. For $\mathcal{ALC}$ and its extensions, it is usually one exponential higher or even undecidable [48][74].

[3]In $DL\text{-}Lite_{horn}$, concept inclusions are restricted to the form $\sqcap_k B_k \sqsubseteq B$, where B and the B_k are concepts.

[4]The language of $DL\text{-}Lite_{bool}$ has object names $a_i (i = 1 \cdots n)$, concept names $A_j (j = 1 \cdots m)$,

[6][49][50] study the problem of importing ontology by queries, which is a variant case of partial ontology reuse where physical access to the imported ontology $\mathcal{O}$ is not allowed. [6] shows that no import-by-query algorithm exists even if the importing ontology $\mathcal{P}$ and the imported ontology $\mathcal{O}$ are expressed in the lightweight DL $\mathcal{EL}$ when the query is concept subsumption or $\mathcal{P}$ reuses atomic roles from $\mathcal{O}$.

4.3 A Logical Model of Distributed Ontology Systems

In this section, we describe the architecture of DOS and present a logic-based definition for DOS.

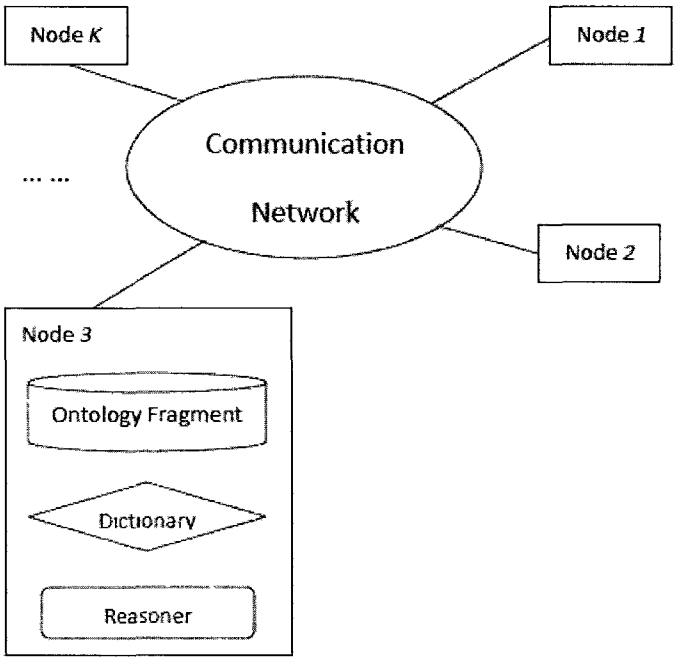

Figure 4.2: Distributed Ontology System Architecture II

In a DOS (see figure 4.2 for architecture), there are a set of autonomous nodes across the network. Each node is equipped with a reasoner so that all the nodes are capable of performing reasoning tasks. A large ontology is divided into smaller pieces

and role names $P_k(k = 1 \cdots l)$. Complex roles R and concepts C are defined as i) $R ::= P_k | P_k^-$; ii) $B ::= \bot | \top | A_j | \geq qR$; iii) $C ::= B | \neg C | C_1 \sqcap C_2$ where B is called basic concept and $q \geq 1$.

(called *ontology fragments*) that are distributed over the nodes, that is, each node has an ontology fragment stored on it. The user can pose queries through any node, but the queries will be evaluated against the whole ontology instead of against the ontology fragment on the node. The ontology fragments are logically interrelated since they belong to the same ontology. Metadata such as relationship between fragments and locations of fragments should be recorded so that it can be used to coordinate the nodes to complete tasks such as query processing, collaborative development of the ontology and so on.

A *master node* is the node where a task is initiated, and the master node is responsible for coordinating other nodes to complete the task. When the task ends, the master node becomes a normal node. That is, master nodes are not fixed in DOS, and a node is a master node only for the time of performing a task that is initiated from itself.

Example 9 *Consider an ontology $\mathcal{O}$ as shown in Table 4.1.*

Table 4.1: Ontology $\mathcal{O}$ in DOS

TBox	$Mother \equiv Woman \sqcap \exists HasChild.Human$ $Father \equiv Man \sqcap \exists HasChild.Human$	
RBox	$HasChild \sqsubseteq AncestorOf$ $Siblings^{-}$	$Trans(AncestorOf)$
ABox	$Human(Vivian)$ $Woman(Ann)$ $HasChild(Ann, Vivian)$ $Siblings(Ann, Wendy)$	$Human(Wendy)$ $Man(Bob)$ $HasChild(Bob, Vivian)$

The axioms in TBox define Mother/Father as woman/man who has a human being child. The axioms in RBox specify properties of roles, HasChild is a subrole of AncestorOf, role AncestorOf is transitive, and role Siblings is an inverse role. The facts in ABox say that Vivian and Wendy are human, Ann is a woman and Bob is a man, Ann/Bob has a child Vivian, Ann and Wendy are siblings.

We divide $\mathcal{O}$ into two fragments F_1 and F_2 as shown in Table 4.2 and 4.3, respectively.

Table 4.2: Ontology Fragment F_1 in DOS

TBox	$Mother \equiv Woman \sqcap \exists HasChild.Human$ $Father \equiv Man \sqcap \exists HasChild.Human$	
RBox	$HasChild \sqsubseteq AncestorOf$ $Siblings^-$	$Trans(AncestorOf)$
ABox	$Woman(Ann)$ $HasChild(Ann, Vivian)$	$Human(Vivian)$

Table 4.3: Ontology Fragment F_2 in DOS

TBox	$Mother \equiv Woman \sqcap \exists HasChild.Human$ $Father \equiv Man \sqcap \exists HasChild.Human$	
RBox	$HasChild \sqsubseteq AncestorOf$ $Siblings^-$	$Trans(AncestorOf)$
ABox	$Man(Bob)$ $HasChild(Bob, Vivian)$	$Human(Wendy)$ $Siblings(Ann, Wendy)$

Here, F_1 and F_2 have the same TBox and RBox, but different ABox. We distribute F_1 and F_2 to node N_1 and N_2, respectively. The dictionaries stored on N_1 and N_2 should contain the properties of each fragment, for instance, F_1 and F_2 contains complete TBox and RBox, incomplete ABox.

When a query, for example,

$$q(x) := Mother(x) \wedge Siblings(x, Wendy)$$

is posed over node N_1, N_1 becomes a master node. N_1 coordinates N_2 in processing query q and returns the answer "Ann" to the user.

Formally,

Definition 2 *A distributed ontology system $\mathcal{I}$ is a triple $< \mathcal{O}, \mathcal{F}, \mathcal{D} >$ where*

1. *$\mathcal{O}$ is an ontology, expressed in a language $\mathcal{L}_\mathcal{O}$ over an signature $Sig(\mathcal{O})$ containing concepts, roles and individuals of $\mathcal{O}$.*

2. *$\mathcal{F}$ is a set of ontology fragments $F_1, \cdots, F_n$ into which $\mathcal{O}$ is divided. Note that ontology fragments are not necessarily disjoint. We denote with $Sig(F_i)$ the signature containing concepts, roles and individuals of ontology fragment F_i. Here, $Sig(F_i) \subseteq Sig(\mathcal{O})$, and $Sig(\mathcal{O}) = \bigcup_{i=1,\ ,n} Sig(F_i)$.*

3. *$\mathcal{D}$ is a set of mappings of the forms:*

$$Sig(\mathcal{S}) \rightsquigarrow \mathcal{F}_i$$

$$F_i \rightsquigarrow F_j$$

where $i, j = 1, \cdots, n$, $Sig(\mathcal{S})$ is a subset of $Sig(\mathcal{O})$. Intuitively, $Sig(\mathcal{S}) \rightsquigarrow \mathcal{F}_i$ specifies relationships between concepts, roles or individuals in $\mathcal{O}$ and fragments, $F_i \rightsquigarrow F_j$ specifies relationships between fragments.

As for the semantics of a distributed ontology system $\mathcal{I}$, we illustrate it from two directions. First, we start from a model $M_{\mathcal{O}}$ of the ontology $\mathcal{O}$, i.e., an interpretation that satisfies the axioms in $\mathcal{O}$. From the definition, we know that axioms in $F_i(i = 1, \cdots, n)$ are subsets of $\mathcal{O}$, thus, $M_{\mathcal{O}}$ is also a model for all ontology fragments F_i. On the other hand, given a model M of an ontology fragment F, there are only two cases that make M a model of $\mathcal{O}$: (1) $\mathcal{O}$ and F coincide, (2) M is a model for all ontology fragments $F_i(i = 1, \cdots, n)$.

DOS has a number of advantages over centralized ontology systems.

- First of all, DOS provides transparency of fragmentation. During the phase of system design, the system administrator should decide how to divide the ontology. According to the applications building on top of the ontology, the system administrator can determine whether a fragment contains complete/incomplete TBox, RBox and ABox. The user does not participate in ontology fragmentation, so they interact with DOS in the same way that they interact with centralized ontology systems.

- Second, DOS promises more flexible replication policy. Distributing replicated data across the machines on a network helps the performance since diverse and conflicting user requirements can be more easily accommodated. DOS provides more options in the number of replicas and the location where they are placed. The system administrator can make the replication policy according to the applications. For example, the ontology fragment that is commonly accessed by one user can be placed on that user's local machine as well as on the machine of another user with the same access requirement.

- Thirdly, DOS is more reliable than centralized ontology systems. In DOS, the failure of a single site or the failure of a communication link is not sufficient to bring down the entire system. Therefore, users can still access other parts of the ontology. Furthermore, with proper replication policies, the ontology fragments on those failed sites maybe replaced by their replicas.

- The fourth advantage of DOS is scalability. It is easier and more economical to expand the system in DOS than in centralized ontology systems. In a distributed environment, it is much easier to accommodate increasing data size. System overhaul is seldom necessary, expansion can be handled by adding processing and storage power to the network. In addition, it usually costs much less to put together a system of small computers than to use the equivalent power of a single big machine.

Furthermore, it is possible that in distributed mode the system efficiency in querying processing can be improved. In DOS, the performance of query processing should be improved if the queries are evaluated against one or several fragments instead of the whole ontology as the knowledge base needs to be accessed becomes smaller.

On the other hand, there are some challenges brought by DOS.

- One is query answering in DOS. Querying ontologies is actually reasoning with

ontologies. In a centralized ontology system, the returned answers are guaranteed to be complete since the ontology is complete. However, in DOS we are dealing with ontology fragments which contain only incomplete pieces of the ontology. Therefore, we need algorithms that can return complete query answers.

- The other challenge is synchronizing replicas of ontology fragments. Since there may exist more than one copy of a fragment on the network, if one of them is edited by a user, then the others should be updated accordingly.

4.4 Modular Ontology Fragments

We distinguish a special case of ontology fragments: *modular fragment* $\mathcal{F}$ of ontology $\mathcal{O}$ where for a query q, the answers to q can be returned by evaluating q against $\mathcal{F}$ instead of $\mathcal{O}$. By Definition 1 in Section 4.2.2, $\mathcal{F}$ is a module of $\mathcal{O}$ with respect to the signature that contains concepts, roles and individuals in q, $\mathcal{O}$ is a conservative extension of $\mathcal{F}$.

Example 10 *Consider the ontology $\mathcal{O}$ in Example 9, we now divide $\mathcal{O}$ into two fragments F_1' and F_2' as shown in Table 4.4 and 4.5, respectively.*

Table 4.4: Modular Ontology Fragment F_1'

TBox	$Mother \equiv Woman \sqcap \exists HasChild.Human$ $Father \equiv Man \sqcap \exists HasChild.Human$	
RBox	$HasChild \sqsubseteq AncestorOf$ $Siblings^-$	$Trans(AncestorOf)$
ABox	$Woman(Ann)$ $HasChild(Ann, Vivian)$	$Man(Bob)$

Consider a query

$$q(x) := Woman(x)$$

Table 4.5: Modular Ontology Fragment F_2'

TBox	$Mother \equiv Woman \sqcap \exists HasChild.Human$ $Father \equiv Man \sqcap \exists HasChild.Human$	
RBox	$HasChild \sqsubseteq AncestorOf$ $Siblings^-$	$Trans(AncestorOf)$
ABox	$Human(Vivian)$ $HasChild(Bob, Vivian)$	$Human(Wendy)$ $Siblings(Ann, Wendy)$

Evaluating q against fragment F_1' returns the same answer set as evaluating q against the whole ontology $\mathcal{O}$. We say fragment F_1' is modular with respect to the signature $\{Woman\}$.

Consider another query

$$q'(x) := Human(x)$$

Evaluating q' against fragment F_2' returns the same answer set as evaluating q' against the whole ontology $\mathcal{O}$. We say fragment F_2' is modular with respect to the signature $\{Human\}$.

We compare the advantages and disadvantages of modular fragments and non-modular fragments in DOS.

- At the stage of fragmentation, in the modular case, algorithms are needed to identify the modular sections within the ontology. Identifying modules within an ontology is known to be computationally expensive from the research of ontology modularization (See related work 4.2.2). In the case of non-modular fragments, fragmentation is easier since we can divide the ontology arbitrarily.

- For query processing, modular fragments are very efficient since only the node containing the modular fragment needs to be accessed, and the queries are evaluated against only a subset of the ontology. In the case of non-modular fragments, all the nodes in the network might need to be accessed, and the messages transported between nodes should be considered.

- As for ontology editing, modular structure is fragile and very subject to collapse, e.g., inserting fact $Human(Fred)$ into fragment F_1' makes the fragment F_2' no longer modular with respect to $\{Human\}$. To re-generate and re-distribute modular fragments require significant work and computationally expensive. On the other hand, non-modular fragments are flexible because we do not need to worry about synchronization between fragments for the purpose of maintaining modular structure. However, if there are replicas of fragments in the network, synchronization between replicas should be considered in both modular and non-modular cases.

4.5 Cases of Fragmentation

We consider how to divide an ontology into smaller pieces at the stage of fragmentation. From what fragments contain, we summarize three cases of ontology fragments:

1. Fragments that contain complete axiom boxes(TBox and RBox) and incomplete set of facts(ABox). This case is suitable for ontologies that have small TBoxes and RBoxes, but large ABoxes.

2. Fragments that contain incomplete axiom boxes and complete set of facts. This kind of fragmentation fits the needs for ontologies that have large TBoxes and RBoxes, but small ABoxes.

3. Fragments that contain both incomplete boxes of axioms and incomplete box of facts, which is suitable for ontologies have large sets of both axioms and facts.

In addition, we can also distinguish two kinds of fragments from a semantic point of view:

1. Modular fragments with respect to certain signatures. That is, given a query expressed over the signature, we can evaluate the query against only its modular

fragment.

2. Non-modular fragments, namely, fragments that are not modular with respect to any signature.

Now we discuss query processing and update synchronization for each case of fragmentation. The idea is that i) if one fragment contains a complete set of axioms/facts of the divided ontology, then any updated axioms/facts to other fragments should be propagated to the fragment in order to maintain the completeness, ii) if one fragment is modular with respect to a certain signature, then updating axioms or facts may lead to the propagation of the axioms and facts or re-generation and re-distribution of modular fragments.

Below is an example where fragments contain complete sets of axioms but incomplete sets of facts, and each fragment is modular with respect to a certain signature.

Example 11 *Given an ontology $\mathcal{O}$ as shown in Table 4.6. The axioms define Employee as those who work on a certain project, and Supervisor those who are supervisor of a certain project. The facts specify two projects PA and PB and their related information such as people who are working on each project, the supervisor, work location and budget of each project.*

We divide $\mathcal{O}$ into two fragments F_1 and F_2 as shown in Table 4.7 and 4.8, respectively.

Here, fragments F_1 and F_2 contain the complete set of axioms, but incomplete set of facts in the ABox. F_1 is modular with respect to signature

$$Sig(K_1) = \{Employee\}$$

F_2 is modular with respect to signature

$$Sig(K_2) = \{Supervisor,\ Budget,\ WorkLocation\}$$

Table 4.6: Ontology $\mathcal{O}$ for Fragmentation

TBox & RBox	$Employee \equiv \exists WorkOn.Project$
	$Supervisor \equiv Employee \sqcap \exists SupervisorOf.Project$
ABox	$Project(PA)$
	$Supervisor(Ann)$
	$SupervisorOf(Ann, PA)$
	$Budget(100,000, PA)$
	$WorkLocation(PA, Ottawa)$
	$WorkOn(Bob, PA)$
	$WorkOn(Cathy, PA)$
	$WorkOn(David, PA)$
	$Project(PB)$
	$Supervisor(Emma)$
	$SupervisorOf(Emma, PB)$
	$Budget(150,000, PB)$
	$WorkLocation(PB, NewYork)$
	$WorkOn(Fred, PB)$
	$WorkOn(Glen, PB)$
	$WorkOn(Helen, PB)$

Table 4.7: Ontology Fragment F_1 for Fragmentation

TBox & RBox	$Employee \equiv \exists WorkOn.Project$
	$Supervisor \equiv Employee \sqcap \exists SupervisorOf.Project$
ABox	$Supervisor(Ann)$
	$Supervisor(Emma)$
	$WorkOn(Bob, PA)$
	$WorkOn(Cathy, PA)$
	$WorkOn(David, PA)$
	$WorkOn(Fred, PB)$
	$WorkOn(Glen, PB)$
	$WorkOn(Helen, PB)$

Therefore, queries expressed over $Sig(K_1)$ or $Sig(K_2)$ can be processed by accessing F_1 or F_2, respectively.

Now let's consider the situation when the ontology is edited simultaneously by two users at different nodes. If one user inserts "$Intern \equiv Employee \sqcap Student$" into fragment F_1, then in order to maintain the completeness of axiom set in fragment F_2, the update should be propagated to F_2. If the other user inserts "$Supervisor(Henry)$"

Table 4.8: Ontology Fragment F_2 for Fragmentation

TBox & RBox	$Employee \equiv \exists WorkOn.Project$
	$Supervisor \equiv Employee \sqcap \exists SupervisorOf.Project$
ABox	$Project(PA)$
	$Supervisor(Ann)$
	$SupervisorOf(Ann, PA)$
	$Budget(100,000, PA)$
	$WorkLocation(PA, Ottawa)$
	$Project(PB)$
	$Supervisor(Emma)$
	$SupervisorOf(Emma, PB)$
	$Budget(150,000, PB)$
	$WorkLocation(PB, NewYork)$

into fragment F_2, the new fact should also be propagated to F_1 in order to make sure that fragment F_1 is still modular with respect to $\{Employee\}$. However, if the updated fact does not compromise the modularity of F_1, then the propagation is not needed.

The following is another example where fragments contain incomplete sets of axioms but complete sets of facts, and the modularity of each fragment is not considered.

Example 12 *Consider an ontology $\mathcal{O}'$ as shown in Table 4.9. The axioms define $Woman/Man$ as female/male Human, $Mother/Father$ as woman/man who has human children, $Grandmother/Grandfather$ as woman/man who has children that has become mothers or fathers. The axioms also specify properties of roles such as $HasChild$ is a subrole of $AncestorOf$, role $AncestorOf$ is transitive, and role Siblings is an inverse role. The facts in ABox say that Ann is a mother, Bob is a father, and Ann has a child Vivian.*

We divide $\mathcal{O}'$ into two fragments F_1' and F_2' as shown in Table 4.10 and Table 4.11, respectively.

Here, fragments F_1' and F_2' contain incomplete set of axioms but complete set of facts. Since F_1' and F_2' are not modular fragments, they will participate in processing all the queries posed over ontology $\mathcal{O}'$

Table 4.9: Ontology Fragment $\mathcal{O}'$ for Fragmentation

TBox & RBox	$Woman \equiv Female \sqcap Human$
	$Man \equiv Male \sqcap Human$
	$Mother \equiv Woman \sqcap \exists HasChild.Human$
	$Father \equiv Man \sqcap \exists HasChild.Human$
	$Grandmother \equiv Woman \sqcap (\exists HasChild.Mother \sqcup \exists HasChild.Father)$
	$Grandfather \equiv Man \sqcap (\exists HasChild.Mother \sqcup \exists HasChild.Father)$
	$HasChild \sqsubseteq AncestorOf$
	$Trans(AncestorOf)$
	$Siblings^-$
ABox	$Mother(Ann)$
	$Father(Bob)$
	$HasChild(Ann, Vivian)$

Table 4.10: Ontology Fragment F_1' for Fragmentation

TBox & RBox	$Woman \equiv Female \sqcap Human$
	$Mother \equiv Woman \sqcap \exists HasChild.Human$
	$Grandmother \equiv Woman \sqcap (\exists HasChild.Mother \sqcup \exists HasChild.Father)$
	$HasChild \sqsubseteq AncestorOf$
	$Trans(AncestorOf)$
	$Siblings^-$
ABox	$Mother(Ann)$
	$Father(Bob)$
	$HasChild(Ann, Vivian)$

Table 4.11: Ontology Fragment F_2' for Fragmentation

TBox & RBox	$Man \equiv Male \sqcap Human$
	$Father \equiv Man \sqcap \exists HasChild.Human$
	$Grandfather \equiv Man \sqcap (\exists HasChild.Mother \sqcup \exists HasChild.Father)$
	$HasChild \sqsubseteq AncestorOf$
	$Trans(AncestorOf)$
	$Siblings^-$
ABox	$Mother(Ann)$
	$Father(Bob)$
	$HasChild(Ann, Vivian)$

When F_1' and F_2' are edited at the same time, propagation of updated axioms is not necessary, but propagation of facts is needed in order to maintain the completeness of facts in each ontology fragment.

In summary, Table 4.12 shows cases of ontology fragmentation and the operations that should be performed at the phase of ontology fragmentation and editing, where "√" means that the operations must be done, "×" the operations are not necessary, and "◯" the operations might be needed.

Table 4.12: Cases of Ontology Fragmentation

		Identifying Modular Fragments	Propagating axiom updates	Propagating fact updates
Complete Axioms & Incomplete Facts	Modular	√	√	◯
	Non-modular	×	√	×
Incomplete Axioms & Complete Facts	Modular	√	◯	√
	Non-modular	×	×	√
Incomplete Axioms & Incomplete Facts	Modular	√	◯	◯
	Non-modular	×	×	×

From Table 4.12, we can see that:

1. In the case where ontology fragments contain complete sets of axioms and incomplete sets of facts,

    - If the fragments are modular, then at the stage of fragmentation, algorithms are needed to identify the modular pieces within the ontology. In addition, to maintain the completeness of axioms, any updates to the axioms at one node must be propagated to all the other nodes. Updating facts results in propagation to nodes where modular fragments with respect to the updated fact are stored, or in the worst case, re-generation and re-distribution of modular fragments.

    - If the fragments are not modular, then the only operation that needs to be performed is the propagation of axiom updates.

2. In the case where ontology fragments contain incomplete sets of axioms and complete sets of facts,

- If the fragments are modular, algorithms identifying modular pieces are needed at the time of fragmentation. And any updates to the set of facts should be propagated to other fragments. Updating axioms lead to the propagation to nodes where modular fragments with respect to the updated axioms are stored, or in the worst case, re-generation and re-distribution of modular fragments.
- If the fragments are not modular, then the only operation that needs to be performed is the propagation of updates related to the facts.

3. In the case where ontology fragments contain incomplete sets of axioms and incomplete sets of facts,
   - If the fragments are modular, algorithms identifying modular pieces are needed at the time of fragmentation. Any updates to the ontology may result in propagation to nodes where modular fragments are stored, or in the worst case, re-generation and re-distribution of modular fragments.
   - If the fragments are not modular, none of the three operations needs to be performed.

4.6 Summary

In this chapter, we formally defined DOS by describing three key components of each node: an ontology fragment, a reasoner and a dictionary containing metadata. Then, we discussed the advantages and challenges of DOS by comparing DOS and centralized ontology systems. We also analyzed cases of ontology fragmentation and discussed the advantages and challenges of each case in terms of query processing and ontology development in DOS.

Chapter 5

Reasoning Problems in Distributed Ontology Systems

5.1 Introduction

Query processing is an important service provided by ontology systems. Unlike in database systems where a query searches information within the content of a database, querying ontologies is essentially reasoning with ontologies. Queries in ontology systems return not only explicit knowledge in ontologies, but also implicit knowledge entailed by the explicit ones. We illustrate the difference using the following example.

Example 13 *Given a database $\mathcal{D}$ containing employee names and their IDs as shown in Table 5.1, and an ontology $\mathcal{O}$ containing information about employee and supervisor as shown in Table 5.2.*

Table 5.1: Database $\mathcal{D}$

Employee	Name	ID
	Ann	1001
	Bob	1002
	Cathy	1003
	David	1004

Table 5.2: Ontology $\mathcal{O}$ for Queries

TBox	$Supervisor \equiv Employee \sqcap \exists SupervisorOf.Project$
ABox	$Supervisor(Ann)$
	$Employee(Bob)$
	$Employee(Cathy)$
	$Supervisor(David)$

Consider queries asking for employee names, i.e., evaluating query

$$q(x) := \exists y\ employee(x, y)$$

against $\mathcal{D}$ *and*

$$q'(x) := employee(x)$$

against $\mathcal{O}$*. Both queries return the same set of answers:*

$$\{Ann, Bob, Cathy, David\}$$

All the employee names returned by $\mathcal{D}$ *can be observed in table employee 5.1. However, in* $\mathcal{O}$ *only Bob and Cathy are explicitly described as employees, Ann and David are inferred by using the axiom in TBox, i.e.,* $Employee(Ann)$ *and* $Employee(David)$ *are implicit knowledge in* $\mathcal{O}$*.*

In this chapter, we discuss three reasoning problems in DOS: i) conjunctive query answering, ii) consistency checking introduced by updating ABoxes, iii) instance realization. We concentrate on the case where ontologies have large ABoxes. We assume that each node is equipped with a DL reasoner, each ontology fragment is non-modular and contains complete set of axioms and incomplete set of facts.

We propose a query processing procedure for conjunctive queries. The main idea is that given a query, the master node (the node that receives the given query) determines facts that involve in evaluating the query by unfolding the query with its sub-concepts and sub-roles. Then the master node collects the involved facts from other nodes and constructs a new ABox which together with the local TBox and RBox forms a new ontology fragment. Finally, the unfolded query is evaluated against the new fragment and the answers are returned to the user. We prove that the procedure returns sound answers for OWL ontologies despite the expressivity of the ontology, and returns complete answers for $\mathcal{SHI}$ given the underlying DL

reasoner at the master node is capable for $\mathcal{SHI}$ ontologies. We also show how to deal with consistency checking introduced by updating ABoxes and instance realization by taking advantage of the query processing procedure.

5.2 Related Work

We discuss three threads of research in this section: DL reasoners, some reasoning algorithms, and works in combining relational databases and ontologies.

5.2.1 DL Reasoners

A reasoner is able to infer logical consequences from a set of asserted facts and axioms. Currently, there exist a number of DL reasoners that are used for real world applications. We briefly introduce some of them as follows.

CEL[1][18] is a free LISP-based reasoner for EL+. It implements a refined version of a known polynomial-time classification algorithm and supports new features like module extraction and axiom pinpointing [19].

RacerPro[2][53] is a commercial LISP-based reasoner for $\mathcal{SHIQ}$ with simple datatypes (i.e., for OWL-DL with qualified number restrictions, but without nominals). It implements a tableau-based decision procedure for general TBoxes and ABoxes.

QuOnto[3][1] is a free Java-based reasoner for DL-lite. It implements a query rewriting algorithm for both consistency checking and query answering for unions of conjunctive queries over DL-Lite knowledge bases, whose ABoxes are managed through relational database technology.

Pellet[4][100] is a free open-source Java-based reasoner for $\mathcal{SROIQ}$ with simple datatypes (i.e., for OWL 1.1). It implements a tableau-based decision procedure for

[1]http://lat.inf.tu-dresden.de/systems/cel/
[2]http://www.racer-systems.com/products/index.phtml
[3]http://www.dis.uniroma1.it/ quonto/
[4]http://www.mindswap.org/2003/pellet/index.shtml

general TBoxes and ABoxes.

KAON2[5][84] is a free Java-based reasoner for $\mathcal{SHIQ}$ extended with the DL-safe fragment of SWRL[6]. It implements a resolution-based decision procedure for general TBoxes and ABoxes.

HermiT[7][82] is a free Java-based reasoner. It works for OWL 2 and supports description graphs. It implements a hypertableau-based decision procedure.

FaCT++[8][110] is a free open-source C++-based reasoner for OWL 2. It implements a tableau-based decision procedure for general TBoxes and ABoxes.

TrOWL[9][112] is a free open-source tractable reasoning infrastructure for OWL 2. It implements approximate reasoning for standard TBox and ABox reasoning, as well as conjunctive query answering for QL, EL and DL.

5.2.2 Tableau Algorithms

In the area of Description Logics, the first tableau-based algorithm was presented by Schmidt-Schauß and Smolka in 1991 for checking satisfiability of $\mathcal{ALC}$-concepts [104]. Since then, this approach has been employed to obtain sound and complete satisfiability and subsumption algorithms for a great variety of DLs extending $\mathcal{ALC}$ [22][23]. Tableau algorithms provide a set of rules for constructing a finite model for a DL concept description. If such model can be constructed successfully, then the given concept description is satisfiable, otherwise, the concept description is unsatisfiable.

Tableau algorithms become inefficient when the size of the model is large or there are many models that need to be examined. The size of the model is determined by the size of the given concept description and existential quantifiers in it. The number of models is determined by disjunction which leads to non-deterministic cases.

[5]http://kaon2.semanticweb.org
[6]http://www.w3.org/Submission/SWRL/
[7]http://hermit-reasoner.com/
[8]http://owl.man.ac.uk/factplusplus/
[9]http://trowl.eu/

Numerous optimizations have been developed in an effort to increase the efficiency of tableau algorithms[10][61][109]. Among them, hypertableau algorithms describe a blocking condition to ensure termination for reducing the nondeterminism due to general inclusion axioms ($C \sqsubseteq D$) [82] [83].

5.2.3 Relational Databases and Ontologies

There exists similarity in the foundation between the schema languages used in relational databases and OWL ontologies. The former is based on first-order logic, and the latter is based on description logics which are fragments of the first-order logic. [76] tries to bridge the gap between OWL and relational databases by extending OWL with integrity constraints similar to that in relational databases. They show that if the integrity constraints are satisfied, they need not to be considered while answering a broad range of positive queries.

In practice, relational database techniques and OWL reasoners are usually combined to provide more efficient reasoning, especially over ontologies with large ABoxes. [57] proposes instance store where instances are kept in a relational database and a reasoner is used to infer ontological information about the classes they belong to. The instance store only works for role-free ABoxes and supports instance retrieval queries.

Another example of this hybrid architecture is SHER [40][34][35] from IBM. [40][34] describe summarization and refinement techniques which are used to perform sound and complete reasoning on large ABoxes stored in relational databases. The procedure of summarization and refinement generates a summary ABox where an individual in it is mapped into a subset of individuals in the original ABox. Then a standard tableau algorithm is applied to the summary ABox whose size is now much smaller than the original one. SHER supports instance retrieval and conjunctive query answering for $\mathcal{SHIN}$ ontologies.

5.3 Distributed Conjunctive Query Answering

In this section, we present a query processing procedure for conjunctive queries in DOS. We focus on ontologies with large ABoxes, therefore, we assume that after fragmentation each ontology fragment is non-modular and contains a complete TBox, RBox and an incomplete ABox. Note that for the query processing procedure, it is not necessary to distinguish TBox and RBox. Therefore, we view TBox and RBox as one box containing axioms.

Recall the general form of conjunctive queries (formula 2.1 in Section 2.2.2):

$$A(\bar{x}) \leftarrow B_1(\bar{y_1}), \cdots, B_n(\bar{y_n})$$

where $A(\bar{x})$ and $B_i(\bar{y_i})(i = 1, \cdots, n)$ are concept or role names in a DL language, A and B_i are DL predicates, $\bar{x}$ and $\bar{y_i}$ are lists of variables and constants that match the arity, i.e., number of attributes of x, y_i respectively.

The idea of the query processing procedure is to construct a new fragment by extracting facts that are involved in query answering from all the fragments, at the same time generate a new query that is logically equivalent to the given query, then evaluate the new query against the new fragment. Specifically, given a query q posed over an ontology $\mathcal{O}$, the master node (the node that receives the q) starts the procedure. The procedure first composes a new query q' by recursively appending sub-concepts(sub-roles) to their parent concepts(roles) in q with disjunction($\sqcup$). q and q' are logically equivalent with respect to $\mathcal{O}$'s TBox and RBox. In addition to q', the procedure also generates a set of instance or role retrieval queries Q that are used to retrieve the facts which involve in answering q(or q'). Then the master node forwards Q to other nodes. All nodes, including the master node, run the queries in Q using their underlying DL reasoners and return the answers to the master node. With the received answers, the master node constructs a new ontology fragment (that is, the set of facts extracted by Q +TBox+RBox). Finally, the master node evaluates q' against the new fragment

using its own reasoner and returns the answers to the user.

Let q be a query posed over an ontology $\mathcal{O}$, C and D concepts in $\mathcal{O}$, r and p roles in $\mathcal{O}$, and Q initially an empty set that will be used to contain fact retrieval queries, the query processing procedure $\mathbb{P}$ for q includes four steps:

1. *Normalize TBox and RBox of $\mathcal{O}$.*

   Input: TBox and RBox in $\mathcal{O}$, initialized Q.

   Output: Normalized TBox and RBox, Q.

   (a) For each axiom of the form $\exists r.C$ or $\forall r.C$, $Q = Q \cup \{r(X,Y), C(X)\}$.

   (b) Replace equivalence with subsumption, that is, replace each axiom of the form $C \equiv D(r \equiv p)$ with $C \sqsubseteq D(r \sqsubseteq p)$ and $D \sqsubseteq C(p \sqsubseteq r)$.

   (c) Transform each general inclusion axiom into its normal form, that is, replace each axiom of the form $C \sqsubseteq D(r \sqsubseteq p)$ with $\top \sqsubseteq \neg C \sqcup D(\top \sqsubseteq \neg r \sqcup p)$.

   (d) Transform the right hand side of each general inclusion axiom into disjunctive normal form.

2. *Construct q' and Q by unfolding q with axioms in the normalized TBox and RBox.*

   Input: q, Q, ontology fragments $F_1, \cdots, F_n$.

   Output: q', Q.

   q' is constructed by doing the following recursively. For each non-atomic concept C or role r in q, search the normalized TBox(RBox). If C or r appears as a conjunctive clause on the right hand side of the inclusion axioms, take the rest of the formula on the right hand side, negate it (which is the sub-concept/sub-role of C/r), then append the result to C or r with disjunction. The recursion ends when all the concepts and roles in the appended clause are atomic. Then $Q = Q \cup \{terms\ in\ q'\}$ where terms are concept or role names.

An algorithm for step 2 is depicted in Table 5.3. The input is the given query q and a set of ontology fragments $\{F_1, \cdots, F_n\}$ whose union is ontology $\mathcal{O}$. After a recursive procedure, the algorithm outputs the unfolded query q' and a set of queries Q that extracts facts from ontology fragments.

3. *Construct a new ontology fragment F'.*

   Input: Q, ontology fragments $F_1, \cdots, F_n$.

   Output: F'.

   Forward Q to other nodes. Each node runs the queries in Q using its underlying reasoner and sends the results to the master node. The master node collects all the instances and combines them with its TBox and RBox to form a new ontology fragment F'.

4. *The master node evaluates q' against F' using its own reasoner and returns the answers to the user.*

   Input: q', F'.

   Output: answers to $q'(q)$.

We illustrate the query processing procedure $\mathbb{P}$ using the following example.

Example 14 *Given an ontology $\mathcal{O}$ about relationships between family members as shown in Table 5.4. we divide $\mathcal{O}$ into two fragments F_1 and F_2 as shown in Table 5.5 and 5.6, respectively. We assume that F_1 and F_2 are located on nodes N_1 and N_2, respectively.*

We consider the following query posed at node N_1.

$$q(x) := Mother(x) \wedge AncestorOf(x, Vivian) \wedge Siblings(Wendy, x)$$

Receiving q, N_1 runs the procedure $\mathbb{P}$:

Table 5.3: Algorithm for Step 2 in Procedure $\mathbb{P}$

Input: q, Q, $\{F_1, \cdots, F_n\}$
Output: q', Q
begin
  for each term a in q **do**
    case of a
      a is a concept:
        begin
        $C := a$
        $a' := a$
        while each term in C is non-primitive **do**
          begin
          search normalized TBox
          if C is a conjunctive clause on the right hand side of the inclusion axioms
          begin
          f := the rest of the formula on the right hand side
          $a' := a' \vee \neg f$
          $C := f$
          end
          end
        end-while
        $a := a'$
        end
      a is a role:
        begin
        $r := a$
        $r' := a$
        while each term in r has subclasses r'
          begin
          search normalized RBox
          if r is a conjunctive clause on the right hand side of the inclusion axioms
          begin
          f := the rest of the formula on the right hand side
          $a' := a' \vee \neg f$
          $r := f$
          end
          end
        end-while
        $a := a'$
        end
    end-case
  end-for
  $q' := q$
  $Q := Q \cup \{$ terms in $q'\}$
  output $\leftarrow q', Q$
end

Table 5 4: Ontology $\mathcal{O}$ for Conjunctive Query Processing

TBox & RBox	$Mother \equiv Woman \sqcap \exists HasChild.Human$
	$Father \equiv Man \sqcap \exists HasChild.Human$
	$Grandmother \equiv Woman \sqcap (\exists HasChild.Mother \sqcup \exists HasChild\ Father)$
	$Grandfather \equiv Man \sqcap (\exists HasChild.Mother \sqcup \exists HasChild.Father)$
	$HasChild \sqsubseteq AncestorOf$
	$Trans(AncestorOf)$
	$Siblings^{-}$
ABox	$Woman(Ann)$
	$Man(Bob)$
	$Grandmother(Cathy)$
	$Grandfather(David)$
	$HasChild(Ann, Vivian)$
	$HasChild(Bob, Vivian)$
	$Siblings(Ann, Wendy)$
	$Human(Vivian)$
	$Human(Wendy)$

Table 5.5. Ontology Fragment F_1 for Conjunctive Query Processing

TBox & RBox	$Mother \equiv Woman \sqcap \exists HasChild.Human$
	$Father \equiv Man \sqcap \exists HasChild\ Human$
	$Grandmother \equiv Woman \sqcap (\exists HasChild.Mother \sqcup \exists HasChild.Father)$
	$Grandfather \equiv Man \sqcap (\exists HasChild.Mother \sqcup \exists HasChild.Father)$
	$HasChild \sqsubseteq AncestorOf$
	$Trans(AncestorOf)$
	$Siblings^{-}$
ABox	$Woman(Ann)$
	$Grandmother(Cathy)$
	$HasChild(Ann, Vivian)$
	$Human(Vivian)$

- ***Step 1***: *Normalize TBox and RBox.*

  1. *From* $\exists HasChild.Human$, $\exists HasChild.Mother$ *and* $\exists HasChild.Father$, *we get* $Q = \{HasChild(x, y), Human(x), Mother(x), Father(x)\}$.

  2. *Replace equivalence with subsumption, we get*

$$Mother \sqsubseteq Woman \sqcap \exists HasChild.Human$$

$$Woman \sqcap \exists HasChild\ Human \sqsubseteq Mother$$

Table 5.6: Ontology Fragment F_2 for Conjunctive Query Processing

TBox & RBox	$Mother \equiv Woman \sqcap \exists HasChild.Human$
	$Father \equiv Man \sqcap \exists HasChild.Human$
	$Grandmother \equiv Woman \sqcap (\exists HasChild.Mother \sqcup \exists HasChild.Father)$
	$Grandfather \equiv Man \sqcap (\exists HasChild.Mother \sqcup \exists HasChild.Father)$
	$HasChild \sqsubseteq AncestorOf$
	$Trans(AncestorOf)$
	$Siblings^-$
ABox	$Man(Bob)$
	$Grandfather(David)$
	$HasChild(Bob, Vivian)$
	$Siblings(Ann, Wendy)$
	$Human(Wendy)$

$$Father \sqsubseteq Man \sqcap \exists HasChild.Human$$

$$Man \sqcap \exists HasChild\ Human \sqsubseteq Father$$

$$Grandmother \sqsubseteq Woman \sqcap (\exists HasChild.Mother \sqcup \exists HasChild.Father)$$

$$Woman \sqcap (\exists HasChild.Mother \sqcup \exists HasChild.Father) \sqsubseteq Grandmother$$

$$Grandfather \sqsubseteq Man \sqcap (\exists HasChild.Mother \sqcup \exists HasChild.Father)$$

$$Man \sqcap (\exists HasChild\ Mother \sqcup \exists HasChild.Father) \sqsubseteq Grandfather$$

3. *Transform each general inclusion axiom into its normal form. That is,*

$$
\begin{aligned}
\top &\sqsubseteq \neg Mother \sqcup (Woman \sqcap \exists HasChild.Human) \\
\top &\sqsubseteq \neg Woman \sqcup \neg\exists HasChild.Human \sqcup Mother \\
\top &\sqsubseteq \neg Father \sqcup (Man \sqcap \exists HasChild.Human) \\
\top &\sqsubseteq \neg Man \sqcup \neg\exists HasChild.Human \sqcup Father \\
\top &\sqsubseteq \neg Grandmother \\
&\quad \sqcup (Woman \sqcap (\exists HasChild.Mother \sqcup \exists HasChild.Father)) \\
\top &\sqsubseteq Grandmother \sqcup \neg Woman \\
&\quad \sqcup (\neg\exists HasChild.Mother \sqcap \neg\exists HasChild.Father) \\
\top &\sqsubseteq \neg Grandfather \\
&\quad \sqcup (Man \sqcap (\exists HasChild.Mother \sqcup \exists HasChild.Father)) \\
\top &\sqsubseteq Grandfather \sqcup \neg Man \\
&\quad \sqcup (\neg\exists HasChild.Mother \sqcap \neg\exists HasChild.Father) \\
\top &\sqsubseteq \neg HasChild \sqcup AncestorOf
\end{aligned}
$$

4. *Transform the right hand side of each general axiom into disjunctive normal form. The above axioms are already in disjunctive normal form.*

- ***Step 2**: Unfold q using axioms in the normalized TBox and RBox.*

$$
\begin{aligned}
q'(x) \;:=\; & (Mother(x) \vee (Woman(x) \wedge \exists y(HasChild(x,y) \wedge Human(y)))) \\
& \wedge (AncestorOf(x, Vivian) \vee HasChild(x, Vivian)) \\
& \wedge Siblings(Wendy, x)
\end{aligned}
$$

And

$$\begin{aligned} Q &= Q \cup \{Mother(x), Woman(x), Human(x), \\ &\quad HasChild(x,y), AncestorOf(x,y), Siblings(x,y)\} \\ &= \{Mother(x), Father(x), Woman(x), Human(x), \\ &\quad HasChild(x,y), AncestorOf(x,y), Siblings(x,y)\} \end{aligned}$$

- ***Step 3:*** *N_1 runs the queries in Q and gets a set of facts relevant to q:*

$$\{Woman(Ann), HasChild(Ann, Vivian), Human(Vivian)\}$$

N_2 runs the queries in Q and returns to $node_1$ its set of relevant facts:

$$\{HasChild(Bob, Vivian), Siblings(Ann, Wendy), Human(Wendy)\}$$

Now N_1 constructs a new ontology fragment F' as shown in Table 5.7.

Table 5.7: Ontology Fragment F' for Conjunctive Query Processing

TBox & RBox	$Mother \equiv Woman \sqcap \exists HasChild.Human$ $Father \equiv Man \sqcap \exists HasChild\ Human$ $Grandmother \equiv Woman \sqcap (\exists HasChild.Mother \sqcup \exists HasChild.Father)$ $Grandfather \equiv Man \sqcap (\exists HasChild.Mother \sqcup \exists HasChild.Father)$ $HasChild \sqsubseteq AncestorOf$ $Trans(AncestorOf)$ $Siblings^-$
ABox	$Woman(Ann)$ $HasChild(Ann, Vivian)$ $HasChild(Bob, Vivian)$ $Siblings(Ann, Wendy)$ $Huamn(Vivian)$ $Human(Wendy)$

- ***Step 4:*** *N_1 evaluates q' against F' and returns the answers to the user, that is* $\{Ann\}$

The following propositions show that query processing procedure $\mathbb{P}$ always returns sound answers, and returns complete answers for $\mathcal{SHI}$ ontologies.

Proposition 1 *Let q of arity n be a conjunctive query posed over $\mathcal{O}$, let q' be the conjunctive query after step 2 in $\mathbb{P}$ to q. Then q and q' are logically equivalent.*

Proof

Assume $q(\bar{x}) := \bigwedge_{i=1,\ ,n} B_i(\bar{y}_i)$,

i). If $q(\bar{x}) := B(\bar{x})$,

Case 1. If B is an atomic concept or role, then after step 2 in $\mathbb{P}$, $q' = q$.

Case 2. If B is not atomic:

Assume in the normalized TBox/RBox of step 1 in $\mathbb{P}$, there exists an inclusion axiom containing B as a conjunctive clause,

$$\top \sqsubseteq B \sqcup U$$

then, we have

$$q'(\bar{x}) := (B(\bar{x}) \vee \neg U(\bar{x}))$$

Now we prove that $q' \equiv q$.

From $\top \sqsubseteq B \sqcup U$, we get $\neg U \sqsubseteq B$. So if B is true, then $\neg U$ is also true, therefore $B \vee \neg U$ is true; if B is false, then $\neg U$ is also false, therefore $B \vee \neg U$ is false. That is,

$$B \vee \neg U \equiv B$$

$$\therefore q' \equiv q$$

ii). Assume that for $q_1(\bar{x}) := \bigwedge_{i=1,\ ,n} B_i(\bar{y}_i)$, after step 2 in $\mathbb{P}$, $q_1 \equiv q_1'$. Now for $q(\bar{x}) := q_1 \wedge B_{n+1}(\bar{y}_{n+1})$, we check

Case 1. If B_{n+1} is an atomic concept or role, then after step 2 in $\mathbb{P}$,

$$q' = q_1' \wedge B_{n+1}(\bar{y}_{n+1})$$

. From $q_1' \equiv q_1$, we get

$$q' \equiv q_1 \wedge B_{n+1}(\bar{y}_{n+1}) \equiv q$$

Case 2. If B_{n+1} is not atomic:

Assume in the normalized TBox/RBox of step 1 in $\mathbb{P}$, there exists an inclusion axiom containing B_{n+1} as a conjunctive clause,

$$\top \sqsubseteq B_{n+1} \sqcup U$$

then, we have

$$q'(\bar{x}) := q_1' \wedge (B_{n+1}(\bar{y}_{n+1}) \vee \neg U(\bar{y}_{n+1})$$

Now we prove that $q' \equiv q$.

From $\top \sqsubseteq B_{n+1} \sqcup U$, we get $\neg U \sqsubseteq B_{n+1}$. So if B_{n+1} is true, then $\neg U$ is also true, therefore $B_{n+1} \vee \neg U$ is true; if B_{n+1} is false, then $\neg U$ is also false, therefore $B_{n+1} \vee \neg U$ is false. That is,

$$(B_{n+1}(\bar{y}_{n+1}) \vee \neg U(\bar{y}_{n+1}) \equiv B_{n+1}(\bar{y}_{n+1})$$

$$\therefore q' \equiv q_1' \wedge B_{n+1}(\bar{y}_{n+1}) \equiv q_1 \wedge B_{n+1}(\bar{y}_{n+1}) \equiv q$$

Therefore, after step 2 in procedure $\mathbb{P}$, $q' \equiv q$.

□

Proposition 2 *(Soundness) Let $\mathcal{O}$ be an OWL ontology in DOS, q of arity n a conjunctive query posed over $\mathcal{O}$, q' a conjunctive query after applying rules in $\mathbb{P}$ to q, F' an ontology fragment constructed according to the rules in $\mathbb{P}$. Then:*

$$F' \models q'(\bar{a}) \Longrightarrow \mathcal{O} \models q(\bar{a})$$

Here $\bar{a}$ is a tuple in F' that satisfies q'.

Proof

Since querying ontologies is actually reasoning with ontologies, we can view $q'(\bar{a})/q(\bar{a})$ as logical consequence of $F'/\mathcal{O}$.

We assume that there exists a tuple $\bar{b}$ of arity n in an arbitrary model of F' such that

$$F' \models q'(\bar{b}) \qquad \text{and} \qquad \mathcal{O} \not\models q(\bar{b}).$$

From proposition 1, q' and q are logically equivalent, we have

$$F' \models q(\bar{b})$$

From the fact that reasoning in DL knowledge bases is monotonic, and $F' \subseteq \mathcal{O}$, we get

$$\mathcal{O} \models q(\bar{b})$$

Which contradicts our assumption.

□

Proposition 3 *(Completeness) Let $\mathcal{O}$ be an $\mathcal{SHI}$ ontology in DOS, q a conjunctive query posed over $\mathcal{O}$, q' a conjunctive query after applying rules in $\mathbb{P}$ to q, F' an ontology fragment constructed according to $\mathbb{P}$. Then:*

$$\mathcal{O} \models q(\bar{a}) \Longrightarrow F' \models q'(\bar{a})$$

Where $\bar{a}$ is a tuple in $\mathcal{O}$ that satisfies q.

Proof

Let $\mathcal{M}$ be a model of $\mathcal{O}$, $\mathcal{M}'$ a model of F', we need to prove that if a tuple $\bar{a}$ of arity n satisfies q in $\mathcal{M}$, then $\bar{a}$ also satisfies q' in $\mathcal{M}'$.

Assume there exists a tuple $\bar{b}$ in $\mathcal{M}$ such that

$$\mathcal{O} \models q(\bar{b}) \qquad \text{and} \qquad F' \not\models q'(\bar{b}).$$

From the general form of conjunctive query

$$q(\bar{x}) := B_1(\bar{y_1}), \cdots, B_n(\bar{y_n}).$$

Since $q(\bar{b})$ holds in $\mathcal{M}$, the body of the general form $B_1(\bar{b_1}), \cdots, B_n(\bar{b_n})$ also holds in $\mathcal{M}$.

F' contains the same TBox and RBox as $\mathcal{O}$. According to procedure $\mathbb{P}$, the ABox of F' contains assertions of concepts and roles that are used to defined $B_i(i = 1, \cdots, n)$ with logical constructors $\forall r.C$, $\exists r.C$, $\neg$ and $\sqsubseteq$ (extracted by queries in Q), which correspond to $\mathcal{ALC}$ description logic extended with $\mathcal{H}$. Furthermore, F' and $\mathcal{O}$ are based on the same DL reasoner which can handle inverse role $\mathcal{I}$ and transitive role $\mathcal{S}$. Therefore, $B_1(\bar{b_1}), \cdots, B_n(\bar{b_n})$ is a logical consequence of F', i.e.,

$$F' \models q(\bar{b})$$

Since q and q' are logically equivalent, we get

$$F' \models q(\bar{b})'$$

This contradicts our assumption.

□

Corollary 1 *Let $\mathcal{O}$ be an $\mathcal{SHI}$ ontology, F' a fragment constructed by applying rules in $\mathbb{P}$, q a query posted over $\mathcal{O}$ and $Sig(q)$ the signature of q. Then F' is a modular fragment in $\mathcal{O}$ with respect to $Sig(q)$.*

Proof

Since q and q' are logically equivalent, evaluating them against F' would return the same set of answers. From Proposition 2 and 3, we get that evaluating q against F' returns the same answer set as evaluating q against $\mathcal{O}$.

□

We discuss the computational complexity of procedure $\mathbb{P}$ step by step. Step 1 is a syntactical transformation of axioms in the TBox and RBox which can be done in polynomial time in the size of TBox and RBox. Step 2 is recursive with the purpose

to unfold concepts and roles where the size of results can reach exponential in the size of the inputs, that is, step 2 in the worst case is in EXPTIME in the size of TBox. Step 3 and 4 are standard OWL-DL reasoning, which is intractable in the worst case. Therefore, the lower bound of complexity of the procedure is EXPTIME. The change of lower bound from polynomial to EXPTIME shows that even if the query processing procedure is implemented on top of a reasoner in a serialized manner, the system efficiency would not be slowed down dramatically.

We now consider the problem whether the query answering procedure $\mathbb{P}$ works for ontologies that are more expressive than $\mathcal{SHI}$. Recall that in OWL 1, there are three profiles of increasing expressive power: OWL Lite, OWL DL and OWL Full. Among them, OWL Lite is the smallest fragment of DL and corresponds to $\mathcal{SHIF}$, that is, $\mathcal{SHI}$ extended with functional roles. We observe that $\mathbb{P}$ does not guarantee the completeness for OWL Lite. The following counter example is used to prove our claim.

Example 15 *Given an $\mathcal{SHIF}$ ontology $\mathcal{O}$ that is divided into two fragments F_1 and F_2 as shown in Table 5.8 and Table 5.9, respectively.*

Table 5.8: $\mathcal{SHIF}$ Ontology F_1 for Conjunctive Query Processing

TBox & RBox	
ABox	IDandName(1001, Adam Jeffrey) Student(Adam Jeffrey)

Table 5.9: $\mathcal{SHIF}$ Ontology F_2 for Conjunctive Query Processing

TBox & RBox	
ABox	IDandName(1001, Adam K. Jeffrey)

Consider a query q posed at the node containing F_1,

$$q(x) := student(x)$$

After applying $\mathbb{P}$, the system returns the answer $\{Adam\ Jeffrey\}$. However, the answer is incomplete. Since role "IDandName" is functional, "Adam Jeffrey" and "Adam K. Jeffrey" are related with the same ID "1001", these two names should belong to the same person whose ID is "1001". Therefore, the answers to query q are $\{Adam\ Jeffrey,\ Adam\ K.\ Jeffrey\}$

5.4 Consistency Checking Introduced by Updating ABoxes

Given an ontology $\mathcal{O}$, the problem of consistency checking we consider in this section is to find out whether adding a new fact to ABox will result in an inconsistent ABox with respect to TBox and RBox. This can be done by querying the negation of the new fact over $\mathcal{O}$. If the answer is "YES", which means that the negation of the fact holds in $\mathcal{O}$, then adding the fact will result in inconsistency. Otherwise, it is safe to update the ABox.

For example,

Example 16 *Given an ontology O as shown in Table 5.10, we consider whether adding into $\mathcal{O}$ a new fact "$\neg FemaleStudent(Ann)$" will cause inconsistency.*

Table 5.10: Ontology O for Consistency Checking

TBox & RBox	$FemaleStudent \equiv Female \sqcap Student$	*
	$MaleStudent \equiv Male \sqcap Student$	
ABox	$Student(Ann)$	*
	$Female(Ann)$	*
	$Student(Bob)$	
	$Male(Bob)$	

*We pose query "$FemaleStudent(Ann)$" over $\mathcal{O}$. In $\mathcal{O}$, axiom and facts with * imply that Ann is a female student, i.e., "$FemaleStudent(Ann)$". Therefore, "YES" will be returned to the user. That is, adding the fact $\neg Woman(Ann)$ will lead to inconsistency in $\mathcal{O}$.*

5.5 Instance Realization

The problem of instance realization is that given an instance, returns the most specific concept name to which the instance belongs to. Instance realization is closely related to a TBox reasoning problem *classification* which returns the hierarchy of all named concepts, for that the concept hierarchy is used to determine the most specific concept that the given instance belongs.

The completeness of knowledge base is essential in instance realization. A complete TBox is indispensable for generating a complete concept hierarchy, and finding the most specific concept that an instance belongs to requires a complete ABox. In our scenario, TBox is complete in each ontology fragment, which guarantees the concept hierarchy is complete. However, ABox is incomplete in each fragment. As a consequence, if we run instance realization over ontology fragments, the results returned by fragments may not include the most specific one.

Therefore, using the querying processing procedure $\mathbb{P}$ we suggest the following approach for handling instance realization in DOS. Given an instance a posed over an ontology $\mathcal{O}$ in DOS,

1. The master node runs classification over its own ontology fragment to generate a concept hierarchy, and forwards to the other nodes the hierarchy and a.

2. Each node, including the master node, executes instance realization for a over its ontology fragment and returns the resulting concept name to the master node.

3. The master node collects the set of concepts from other nodes and determines the most specific one among them, denoted C. Using the concept hierarchy, the master node searches for the sub-concepts of C.

    - If C has no sub-concepts in the hierarchy. then C is the most specific

concept that a belongs to and C is returned to the user.

- If C has a sub-concept, denoted D, then the master node poses a query asking if a is an instance of D. If the answer is "No", then C is the most specific concept that a belongs to and returned to the user. Otherwise, the master node will check the sub-concepts of D until "No" is returned.

We illustrate this approach using the following examples.

Example 17 *Given an ontology O which is divided into two fragments: F_1 and F_2 as shown in Table 5.11 and 5.12, respectively. We assume that F_1 is located on node N_1 and F_2 on node N_2. We consider instance realization for "Ann" initiated from N_1.*

Table 5.11: Ontology Fragment F_1 for Instance Realization

TBox & RBox	$Woman \equiv Female \sqcap Human$
	$Man \equiv Male \sqcap Human$
	$Mother \equiv Woman \sqcap HasChild.Human$
	$Father \equiv Man \sqcap HasChild.Human$
	$Grandmother \equiv Mother \sqcap (HasChild.Mother \sqcup HasChild.Father)$
	$Grandfather \equiv Father \sqcap (HasChild.Mother \sqcup HasChild.Father)$
ABox	$Female(Ann)$
	$Grandmother(Ann)$

Table 5.12: Ontology Fragment F_2 for Instance Realization

TBox & RBox	$Woman \equiv Female \sqcap Human$
	$Man \equiv Male \sqcap Human$
	$Mother \equiv Woman \sqcap HasChild.Human$
	$Father \equiv Man \sqcap HasChild.Human$
	$Grandmother \equiv Mother \sqcap (HasChild.Mother \sqcup HasChild.Father)$
	$Grandfather \equiv Father \sqcap (HasChild.Mother \sqcup HasChild.Father)$
ABox	$Human(Ann)$

First of all, N_1 generates the concept hierarchy for $\mathcal{O}$, see Figure 5 1, and forwards the hierarchy and "Ann" to node N_2.

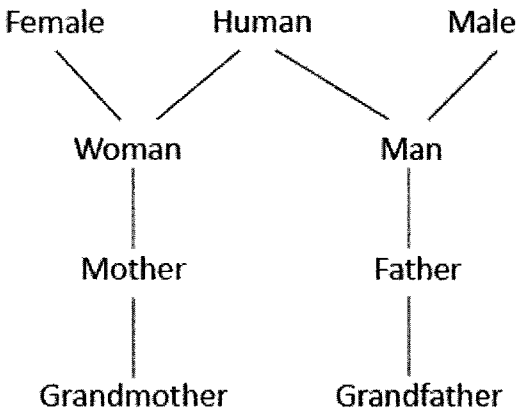

Figure 5.1: Example of Concept Hierarchy

N_1 and N_2 run instance realization for "Ann" over their fragments and get "Grandmother" and "Human", respectively. Then N_1 picks "Grandmother" and checks its sub-concepts. In this case, "Grandmother" does not have any sub-concepts, therefore, "Grandmother" is the most specific concept that "Ann" belongs to and is returned to the user.

The following is a more complex example illustrating how to handle instance realization in DOS.

Example 18 *Given an ontology $\mathcal{O}$ which is divided into two fragments: F_1' and F_2' as shown in Table 5.13 and 5.14, respectively. We assume that F_1' is located on node N_1 and F_2' on node N_2. Again, we consider instance realization for "Ann" initiated from N_1.*

First of all, N_1 generates the concept hierarchy for $\mathcal{O}$, see Figure 5.1, and forwards the hierarchy and "Ann" to node N_2.

N_1 and N_2 run instance realization for "Ann" over their fragments, respectively. Only N_1 returns "Woman". Then N_1 checks sub-concept of "Woman", which is

Table 5.13: Ontology Fragment F_1' for Instance Realization

TBox & RBox	$Woman \equiv Female \sqcap Human$
	$Man \equiv Male \sqcap Human$
	$Mother \equiv Woman \sqcap HasChild.Human$ *
	$Father \equiv Man \sqcap HasChild.Human$
	$Grandmother \equiv Mother \sqcap (HasChild.Mother \sqcup HasChild.Father)$
	$Grandfather \equiv Father \sqcap (HasChild.Mother \sqcup HasChild.Father)$
ABox	$Woman(Ann)$ *
	$HasChild(Ann, Vivian)$ *

Table 5.14: Ontology Fragment F_2' for Instance Realization

TBox & RBox	$Woman \equiv Female \sqcap Human$
	$Man \equiv Male \sqcap Human$
	$Mother \equiv Woman \sqcap HasChild.Human$ *
	$Father \equiv Man \sqcap HasChild.Human$
	$Grandmother \equiv Mother \sqcap (HasChild.Mother \sqcup HasChild.Father)$
	$Grandfather \equiv Father \sqcap (HasChild.Mother \sqcup HasChild.Father)$
ABox	$Human(Vivian)$ *

*"Mother", and poses a query "? Mother(Ann)" over $\mathcal{O}$ asking if "Ann" is an instance of "Mother". In this case, we get "Yes" (from the axiom and facts with *). Then the master node continue checking the sub-concept of "Mother", which is "Grandmother", by posing query "? Grandmother(Ann)" over $\mathcal{O}$, this time, we get "No". Therefore, "Mother" is the most specific concept that "Ann" belongs to and is returned to the user.*

From the above examples, we can see that the content of ontology fragments makes difference in the problem of instance realization in DOS. In the first example, the recursion of checking sub-concepts is executed only once, while in the second example, the recursion is done twice. Using this approach, in the worst case, the master node would need to check a long concept hierarchy chain. An optimization can be done during ontology fragmentation where the instances that belong to the same concept hierarchy chain can be grouped into one fragment.

5.6 Summary

In this chapter, we presented a distributed query processing procedure for conjunctive queries in DOS. We proved that the procedure always returns sound answers for OWL ontologies, and returns complete answers for $\mathcal{SHI}$. We also showed how to handle consistency checking introduced by updating ABoxes and instance realization in DOS by using the query processing procedure for conjunctive queries.

Chapter 6

Collaborative Ontology Development in Distributed Ontology Systems

In this chapter, we discuss a prototype for supporting synchronous and non-locking editing in DOS.

6.1 Introduction

While ontologies may be designed by an individual, many are created in a collaborative manner involving dozens of individuals with varying backgrounds and technical expertise. With the general acceptance, users of ontology systems have become quite familiar and comfortable with the concept of user-contributed content, both in their personal and professional lives. There has been a strong demand for tools that would support collaborative ontology development [65][102].

In collaborative ontology development, we consider two modes of editing:

1. Synchronous editing. Multiple ontology users simultaneously connect to an ontology stored on a server, and changes to the ontology take effect immediately.

   In this mode of editing, there is only a single master version of ontology at one time, users who are editing the ontology must always be connected to the same server. Disconnected editing is not possible. Changes (and errors) are immediately visible to all editors involved, and there is no environment to test a potentially harmful series of changes. A mistake, conflict, or inconsistency can be disastrous in a synchronous environment. An error could result in the

knowledge base becoming unusable for all connected users and editors.

2. Asynchronous editing. Multiple ontology users work on individual copies of the ontology, and submit their changes to a central server.

   In asynchronous editing, there always exists multiple copies of the ontology. Potentially dangerous changes can be independently developed, verified to be safe and then integrated with the rest of the ontology, thereby reducing the cost associated with fixing errors.

In collaborative ontology development, a conflict may occur when multiple users simultaneously edit the same piece of knowledge. For instance, a user defines a concept C as A while another user defines C as B at the same time. Means of preventing conflicts can be divided into two categories:

1. Locking. A section of the ontology is locked when it is being edited in order to prevent other developers from editing it at the same time.

   The advantage of locking is that it pro-actively prevents conflicts. However, editing becomes more difficult, especially as the number of locks increases and other users are prevented from accessing large portions of the knowledge base. In an asynchronous setting, locking also requires some means of modularizing or segmenting the ontology [101].

2. Non-locking. Locks are not used, conflicts are fixed after they occur.

   With non-locking methods, anyone is able to edit anything at any time. This makes fixing errors and conflicts difficult.

6.2 Related Work

There are some ontology editors that allow asynchronous and locking editing. Using Hozo[1] [69], an ontology can be divided into smaller pieces, called ontology modules, and versions of the ontology and access to it are managed by the server. When a developer edits the a module, he locks the module and edits it in his personal space. The developer is responsible to check the potential conflicts and upload the updated version to the server. In Hozo, when a change is made, information about how this change would affect other concepts or roles will be displayed to help the developer to decide whether continue or abort the change. Similar to Hozo, DILIGENT [113] and ONKI [114] [103] support collaborative development of ontology through shared space for ontologies, but checking potential conflicts is done manually.

ContentCVS[2] [62] uses a client-server architecture to support asynchronous and non-locking collaborative editing. A CVS server stores the current version of a project and its change history; CVS clients connect to the server to export a copy of the ontology, allowing developers to work on their own ""local" copy, and then later to commit their changes to the server. It allows several developers to make changes concurrently to a project, but the server only accepts changes to the latest version of any given project file. Developers should hence use the CVS client to regularly commit their changes and update their local copy with changes made by others. Manual intervention is needed when a conflict arises between a committed version in the server and a yet-uncommitted local version. Conflicts are reported whenever the two compared versions of a file are not equivalent according to a given notion of equivalence between versions of a file.

Collaborative Protégé[3] [111] provides a client-server mode to support synchronous

[1] http://www.hozo.jp
[2] http://krono.act.uji.es/people/Ernesto/contentcvs
[3] http://protege.stanford.edu

and non-locking editing. It has support for different user roles which have different privileges in ontology editing. It also allows users to hold discussions about the ontology components and changes using typed annotations, and tracks the change history of the ontology entities. Information supporting the above functionalities is stored in a system knowledge base, called CHAO (Changes and Annotations ontology) [87].

There exist several wiki-based environments that support editing ontologies and instance data, such as BiomedGT[4] and Cicero [33]. These tools provide a natural forum for discussions, and the provenance information for suggested changes is easy to achieve. However, they are not intended for ontology development and users cannot easily edit class definitions using this kind of framework.

6.3 Requirements for Supporting Collaborative Ontology Development

There are a number of important requirements for ontology development tools that cope with the contradiction of the need for close collaboration and the distribution of developers [9][86][111]. We summarize the requirements as follows:

1. Synchronous or asynchronous access to ontologies. Depending on the size of the group and the complexity of the ontology, users may prefer synchronous or asynchronous editing.

2. Integration of discussions. An ontology reflects an agreement on a domain conceptualization among people in a community, so tools supporting discussion are essential. Discussions should be accessible from the ontology components that are being discussed, and when a user examines or writes a discussion, the ontology components involved in the discussion should also be accessible.

[4]http://biomedgt.nci.nih.gov

3. Tracking provenance of information. With multiple authors contributing to the ontology, it is necessary to keep a history for ontology components so that the users can find out who makes specific changes, when and who proposes a change and so on.

4. Conflict detection. In asynchronous editing, it is preferable if the tools can compare potential conflicting versions and identify conflicting parts.

5. Access control. With most existing tools, any user with writing privileges can edit anything in an ontology. However, we consider that different privileges should be assigned to different users. For example, a user with expertise in an area represented in part of the ontology should be able to edit only the part he is familiar with, but browse the whole ontology.

6. Workflow support. Many collaborative development projects have specific workflows associated with making changes. A workflow specification usually includes different tasks that developers are in charged with, the process for proposing a change and reaching consensus, roles that different developers play and so on.

7. Scalability and reliability. Tools in production systems should consider scalability which deals with increase in both the size of ontology and the number of users, and reliability which deals with safety and availability of the data.

No current ontology editor supports all the requirements listed here, they only support some of these features. For instance, Collaborative Protégé supports integration of discussions, tracking provenance of information, workflow and access control to some extend, and ContentCVS supports integration of discussion and conflict detection.

6.4 Supporting Collaborative Ontology Development

In this section, we discuss how to support synchronous and non-locking collaborative ontology development in DOS in a way similar to that in Collaborative Protégé.

6.4.1 Collaborative Protégé

The key feature of Collaborative Protégé is the ability to create annotations which are typed comments such as example, proposal, question and so on. Annotations are attached to ontology components, or to the descriptions of ontology changes, or to other annotations. The structure of the annotations are defined in the Changes and Annotations ontology (CHAO). CHAO contains classes that define the annotation types and classes for describing different types of changes that a user can make in an editing session. Instances of the annotation types represent actual annotations that users are making on ontology components or changes.

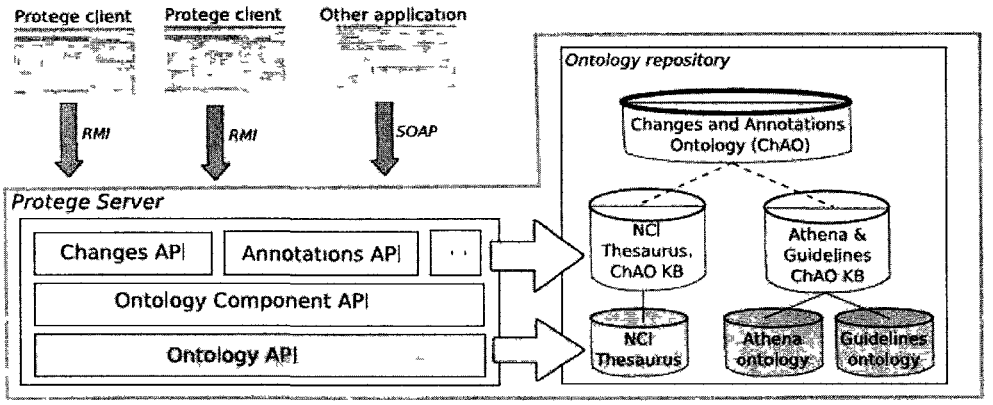

Figure 6.1: Client-Server Architecture of Protégé (Source [111])

Figure 6.1 shows an overview of the main components of Collaborative Protégé. The Protégé server has an ontology repository that contains all the ontologies that Protégé clients can edit in the collaborative mode. The repository has CHAO knowledge bases which are instances of the CHAO classes for each of the domain ontologies

in the repository. These instances represent the changes and the annotations for the corresponding ontology. Several related domain ontologies can share the same CHAO knowledge base.

When a user edits the domain ontology in the Protégé client, each change that the user performs is sent to the server. The server first updates the central (server-side) ontology, then pushes the change to the other clients so that other Protégé users can see them immediately. Finally the server creates one or several CHAO instances that represent the change [87]. When a user creates an annotation in the Protégé client, the Protégé server adds the corresponding instances to the CHAO knowledge base.

6.4.2 Collaborative Ontology Development in DOS

We are aiming at providing a synchronous and non-locking mode for collaborative ontology development in DOS, as well as allowing access control, history tracking and workflow support. Our idea is similar to Collaborative Protégé. CHAO knowledge bases contain annotations and an annotation can be always related to the content of ontology. Therefore, we divide a CHAO knowledge base into fragments and place them on the same nodes where their associated ontology fragments locate. CHAO contains metadata about the structure of annotation, so we keep a copy of it on each node. Since Collaborative Protégé is an ontology editor which supports multiple ontology languages, the "Ontology Components API" provides a meta-language for describing representational entities in different ontology languages. However, in DOS the heterogeneity of ontology language is no longer a problem since there exists only one OWL ontology in the system.

Figure 6.2 shows the architecture of DOS for supporting collaborative ontology development. Each node contains four components: CHAO, an ontology fragment, a CHAO knowledge base fragment in accordance with the ontology fragment and a dictionary keeping the information about each node in the network.

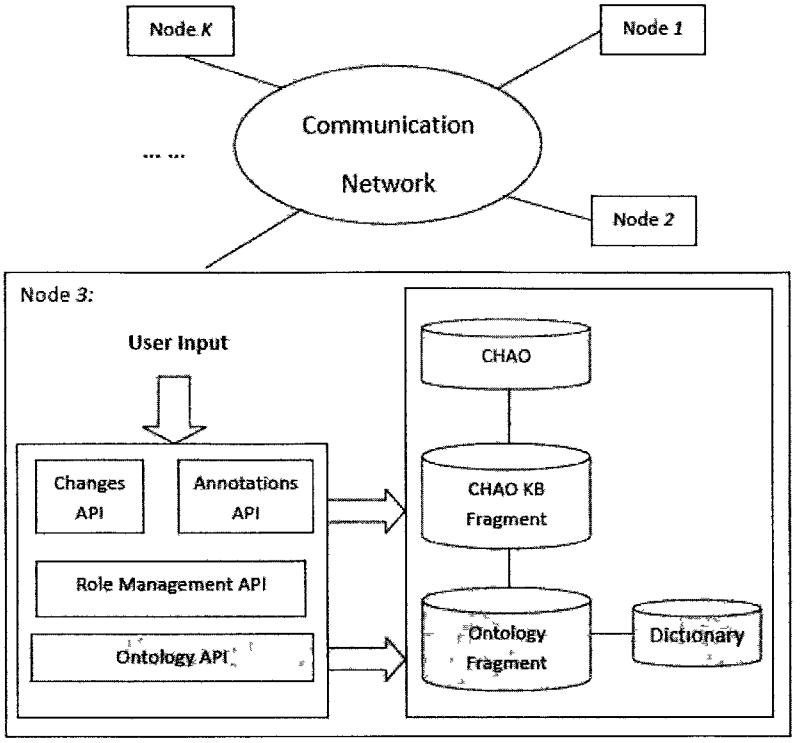

Figure 6.2: DOS Architecture for Collaborative Ontology Development

We now discuss the modules behind APIs in Figure 6.2.

- There exists a Change module behind Change API which provides access to classes representing different types of changes that can occur in an ontology. It contains at least four classes of changes: Axiom Creation, Axiom Deletion, Fact Creation and Fact Deletion. An instance of the classes corresponding to an event that happens in the ontology. For example, an instance of the class Axiom Creation represents a new class creation in TBox or an new axiom in RBox. Users can also define new types of changes in this module.

  When a user performs a change to the ontology, using the information in dictionary, the master node first pushes the change to those ontology fragments that contain the corresponding content, then creates and pushes annotations representing the change to the CHAO knowledge base fragments accordingly.

Figure 6.3 shows a layering scheme for updating changes to an ontology where each layer solves a subproblem.

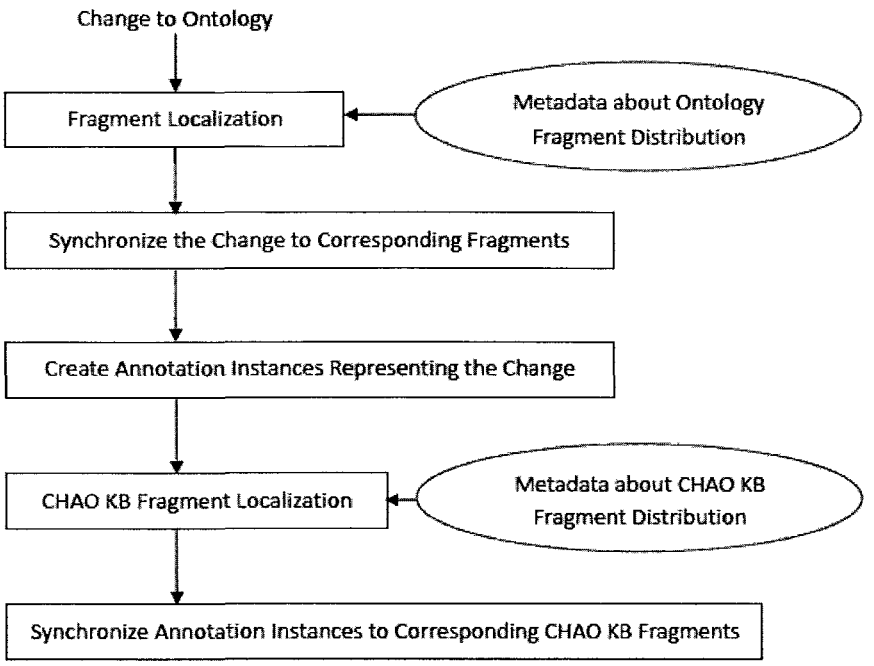

Figure 6.3: Layering Scheme for Updating Changes to Ontology

- The Annotations module contains classes representing different types of annotations including Comment, Question, Advice and Example. An instance of these classes represents an annotation attached to an ontology element, or a change in the ontology, or even another annotation. Users can also create classes for their own types of annotation.

  When a user creates an annotation, the master node synchronizes the corresponding instances to other CHAO knowledge base fragment accordingly, a layering scheme for creating annotations is shown in Figure 6.4.

- The Role Management module provides means for defining roles for users and granting certain privileges to users. It also checks the user credential at login

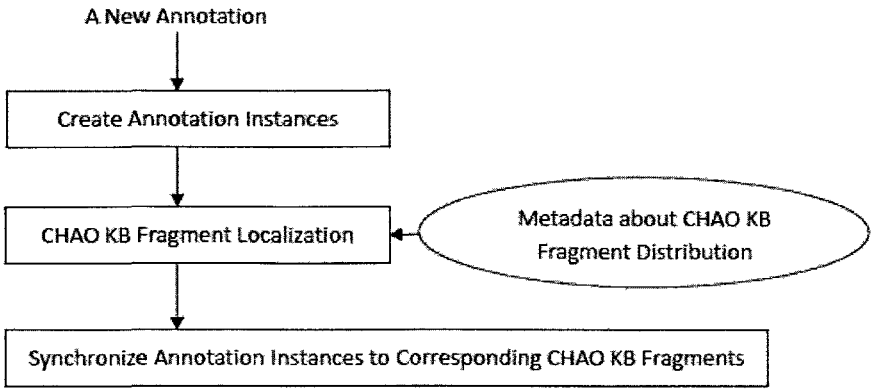

Figure 6.4: Layering Scheme for Creating Annotations

time, and verifies the privileges of each user when a user performs a certain operation to the ontology or to the CHAO knowledge base.

- There is a Workflow module behind the Ontology API which allows users to define workflows for collaborative ontology development. For example, a voting workflow for Fact Deletion can be defined as four phases including proposing fact deletion, voting, counting and deciding. Each phase has some parameters, for example, the number of voting should be greater than ten and so on.

6.5 Summary

In this chapter, we described a prototype for supporting collaborative ontology development in DOS. The distributed architecture promises synchronous and non-locking editing, as well as reliability and scalability for the system. The usage of CHAO ontology allows history tracking for each change made in the interested domain ontology, role management allows access control for users, and user-defined workflows allow flexibility when modifying the domain ontology.

Chapter 7

Conclusion and Future Directions

7.1 Conclusion

In this thesis we are concerned with finding ways to provide reliable and scalable management for large ontologies. We propose distributed ontology systems where large ontologies are divided into fragments and distributed over autonomous nodes across the network.

Essentially, there are three components in each node of DOS: an ontology fragment, a DL reasoner so that each node is capable of reasoning tasks, and a dictionary containing metadata which is used to coordinate other nodes in performing system tasks. Comparing with the systems where ontologies are located on a single computer, DOS is more reliable as the single point of failure, which often happens in centralized systems, can be avoided. DOS is also more scalable since adding processing and storage power to the network is usually easier than setting up a server which is usually responsible for more complex computing tasks.

We also analyze cases of ontology fragmentation. From a syntactic viewpoint, we classify three categories of fragments depending on if the fragment contains complete/incomplete sets of axioms/facts. From a semantic aspect, an ontology fragment can be modular or non-modular with respect to a certain signature. For each case of fragmentation, we discuss advantages and challenges in terms of query processing and update synchronization.

Regarding the problems of querying ontologies in DOS, we first propose a query processing procedure for conjunctive queries. We prove that our method always

returns sound answers for OWL ontologies, and returns complete answers for $\mathcal{SHI}$. Then we show that the problem of consistency checking introduced by updating ABoxes can be reduced to conjunctive queries answering by querying the negation of the updated fact. We also discuss how to deal with instance realization in DOS using the query processing procedure for conjunctive queries.

Finally, we describe a prototype which promises synchronous and non-locking editing for collaborative ontology development in DOS. We discuss the functionality of each module in the prototype and show how to provide access control, history tracking and flexible workflows for users.

7.2 Future Work

Since we are the first to propose a distributed environment for large ontologies, and what we have done in this thesis is just the beginning of research in this area, there are many problems still open. We summarize the future work into four directions.

The first direction is about choosing modular or non-modular fragments during ontology fragmentation. In practice, the strategy deciding modular or non-modular depends on the application. We consider a non-modular case in this thesis, but some applications may become more efficient with modular fragments. For instance, for applications where a set of queries are posed over the ontology with a high frequency, generating a modular fragment with respect to those queries during fragmentation would help in query processing.

The second direction is about querying ontologies. We propose a method for the case where ontologies have large ABoxes and each fragment contains a complete TBox and RBox. However, in some real world applications, ontologies may contain large TBoxes and RBoxes, or even the ABoxes may be very large. These are the cases where our method may be significantly challenged.

The third direction is about the optimization of query processing procedure. We

consider how to minimize the number of ontology fragments that need to be accessed when evaluating queries. We suggest that facts being accessed frequently by queries should be grouped into one ontology fragment during fragmentation. By doing so, less fragments need to be accessed and message exchange between nodes is decreased so that system efficiency can be improved. We also observe that the approach for instance realization can be optimized in a similar way. That is, instances belonging to the concepts that have the relationship of subsumption can be put into the same fragment to cut the message exchange between nodes and number of recursion. Note that it is not necessary for such fragments to be modular, but the fragmentation should be done toward a modular structure.

The last direction is implementation of the query processing procedure. The procedure can be implemented on top of a DL reasoner that is capable at least for $\mathcal{SHI}$. For testing purposes, we suggest to use the ontologies in Tones Ontology Repository[1] which contains over 200 ontologies and about half of them are less expressive than or equivalently expressive to $\mathcal{SHI}$. When comparing system performance with centralized ontology systems, there are three aspects that need to be considered : system response time, computational resources consumed by each node, message and data exchange between nodes in the system. We have to expectations with respect to DOS's performance. One is to handle large ontologies that can not be handle by systems reasoning with centralized ontologies, for example, execute instance realization with yOWL. We also expect DOS can beat centralized systems in response time, especially with optimization techniques mentioned above.

[1]http://owl.cs.manchester.ac.uk/repository/

Bibliography

[1] Acciarri A., Calvanese D., Giacomo D. G., Lembo D., Lenzerini M., Palmieri M. and Rosati R.: QuOnto: Querying ontologies. In Proc. of the 20th National Conference on Artificial Intelligence (AAAI 2005).

[2] Auer, S., Dietzold, S., Riechert, T.: OntoWikiCa tool for social, semantic collaboration. In Cruz, I., Decker S., Allemang D., Preist C., Schwabe D., Mika P., Uschold M., Aroyo L.M. (eds.) ISWC 2006. LNCS, vol. 4273. Springer, Heidelberg.

[3] Bochmann V. G.: Concepts for distributed systems design. Berlin: Springer-Verlag, 1983.

[4] Berners-Lee T.: WWW past & future, 2003. Available at http://www.w3.org/2003/Talks/0922-rsoc-tbl/.

[5] Beneventano D., Bergamaschi S., Castano S., Corni A., Guidetti R., Malvezzi G., Melchiori M., and Vincini M.: Information integration: the MOMIS project demon- stration. In Proc. of the 26th Int. Conf. on Very Large Data Bases (VLDB 2000), 2000.

[6] Baader F., Brandt S., Lutz C.: Pushing the EL envelope. In Proceedings of IJCAI 2005.

[7] Berners-Lee T. and Connolly D.: Notation3(N3): A readable RDF syntax. 2008. Available at http://www.w3.org/TeamSubmission/n3.

[8] Bellifemine F., Caire G. and Greenwood D.: Developing multi-agent systems with JADE. John Wiley and Sons, 2007.

[9] Bao J., Caragea D., Honavar V.: Towards collaborative environments for ontology construction and sharing. In International Symposium on Collaborative Technologies and Systems (CTS 2006). IEEE Press; 2006: 99-108. 2.

[10] Baader F., Calvanese D., McGuinness D., Nardi D. and Patel-Schneider F. P., editors. The description logic handbook: theory, implementation, and applications. In Cambridge University Press, 2003.

[11] Brams S. J. and Fishburn P.C.: Voting procedures. In Handbook of Social Choice and Welfare. Arrow K. J., Sen A. K. and Suzumura K. editors, Volume 1, Chapter 4. Elsevier Science Publishers B.V.: Amsterdam, Netherland, 2002.

[12] Bell D. and Grimson J.: Distributed database systems. Reading Mass.: Addison-Wesley, 1992.

[13] Brickley D., Guha V. R.: RDF vocabulary description language C RDF Schema. http: //www.w3.org/TR/rdf-schema/, 2004.

[14] Brachman R., Hector L.: Knowledge representation and reasoning. Morgan Kaufmann 2004.

[15] Bernstein P., Haas L.: Information Integration in the Enterprise. Communications of the ACM 51(9), 72C79 (2008).

[16] Besnard P. and Hunter A.: Elements of Argumentation. The MIT Press, 2008.

[17] Bock J., Haase P., Ji Q., Volz R.: Benchmarking OWL reasoners. In proceeds of ARea2008 Workshop on Advancing Reasoning on the Web: Scalability and Commonsense at the 5th European Semantic Web Conference (ESWC2008), June 2008, Tenerife, Spain.

[18] Baader, F., Lutz, C., Suntisrivaraporn, B.: CELa polynomial-time reasoner for life science ontologies. In Furbach, U., Shankar, N., eds.: Proceedings of the 3rd International Joint Conference on Automated Reasoning (IJCAR06). Volume 4130 of Lecture Notes in Artificial Intelligence., Springer-Verlag (2006) 287C291.

[19] Baader F., Penaloza r. and Suntisrivaraporn B.: Pinpointing in the Description Logic EL+. In Proceedings of the 30th German Conference on Artificial Intelligence (KI2007), LNAI, Springer-Verlag, 2007.

[20] Rule-based expert systems: the MYCIN experiments of the Stanford Heuristic Programming Project. Edited by Bruce G. Buchanan and Edward H. Shortliffe, 1984. Ebook version available at http://www.aaai.org/AITopics/pmwiki/pmwiki.php/AITopics/RuleBasedExpertSystems.

[21] Blackburn P., Seligman J.: Hybrid languages. Journal of Logic, Language and Information 4:251C272, 1995.

[22] Baader F. and Sattler U.: Tableau algorithms for description logics. In R. Dyckhoff, editor, Proceedings of the International Conference on Automated Reasoning with Tableaux and Related Methods (Tableaux 2000), volume 1847 of Lecture Notes in Artificial Intelligence, pages 1-18, St Andrews, Scotland, UK, 2000. Springer-Verlag.

[23] Baader F. and Sattler U.: An overview of tableau algorithms for description Logics. Studia Logica, 69:5-40, 2001.

[24] Calì A., Calvanese D., De Giacomo G., and Lenzerini M.: Data integration under integrity constraints. In Proc. of the 14th Conf. on Advanced Information Systems Engineering (CAiSE 2002), pp 262-279.

[25] Chen X., Dumontier D.: A framework for distributed ontology systems. In Proc. of the Second Canadian Semantic Web Working Symposium(CSWWS09), page 137-148, 2009.

[26] Calvanese D., Giacomo G. D. and Lenzerini M.: A framework for ontology integration. In Isabel Cruz, Stefan Decker, Jérome Euzenat, and Deborah McGuinness, editors, The Emerging Semantic Web _ Selected Papers from the First Semantic Web Working Symposium, pages 201-214. IOS Press, 2002.

[27] Calvanese D., Giacomo G. D., Lembo D., Lenzerini M., Rosati R.: Tractable reasoning and efficient query answering in description logics: the DL-Lite family. In Journal of Automated Reasoning 9 (2007) 385C429.

[28] Calvanese D., Giacomo G. D., Lenzerini M., Rosati R., and Vetere G.: Logical foundations of peer-to-peer data integration. In Proc. of the 23rd ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2004), pages 241-251.

[29] Cuenca Grau B., Horrocks I., Motik B., Parsia B., Patel-Schneider P., Sattler U.: OWL 2: The next step for OWL. Journal of Web Semantics, 6(4):309-322, November 2008.

[30] Catarci T. and Lenzerini M.: Representing and using interschema knowledge in cooperative information systems. J. of Intelligent and Cooperative Information Systems, 2(4):375-398, 1993.

[31] Coskun G., Luczak-Rösch M., Heese R., Paschke A.: Survey of ontology modularizaion for corporate ontology engineering. In Proc. of the Second Canadian Semantic Web Working Symposium(CSWWS09), page 111-122, 2009.

[32] Dunne. P. E.: Extremal behaviour in multiagent contract negotiation. Journal of AI Research.23:42-78, 2005.

[33] Dellschaft K., Engelbrecht H., Barreto J.M., Rutenbeck S., Staab S.: Cicero: Tracking design rationale in collaborative ontology engineering. In Proceedings of the 5th European Semantic Web Conference (ESWC 2008) Demo Session.

[34] Dolby J., Fokoue A., Kalyanpur A., Kershenbaum A., Schonberg E., Srinivas K. and Ma L.: Scalable semantic retrieval through summarization and refinement. AAAI 2007, page 299-304.

[35] Dolby J., Fokoue A., KalyanpurA., Ma L., Schonberg E., Srinivas K. and Sun X.: Scalable grounded conjunctive query evaluation over large and expressive knowledge bases. In Proc. of International Semantic Web Conference 2008. Page 403-418.

[36] Dou D., McDermott D., and Qi P.: Ontology translation on the semantic web. In International Conference on Ontologies, Databases and Applications of Semantics, 2003.

[37] Serafini L. and Tamilin A.: DRAGO: Distributed reasoning architecture for the semantic web. In Proceeds of the Second European Semantic Web Conference (ESWC'05), Springer-Verlag, 2005.
http://sra.itc.it/projects/drago.

[38] Serafini L. and Tamilin A.: Distributed instance retrieval in heterogeneous ontologies. In Proc. of the Second Italian Semantic Web Workshop Semantic Web Applications and Perspectives (SWAP'05), CEUR Workshop Proceedings, ISSN 1613-0073, 2005.

[39] FIPA Specification Part 2 - agent communication language. Available at http://www.fipa.org/repository/aclspecs.html.

[40] Fokoue A., Kershenbaum A., Ma L., Schonberg E., and Srinivas K.: The summary Abox: cutting ontologies down to size. In Proc. of 5th International Semantic Web Conference, Athens, GA, USA, November 5-9, 2006, LNCS 4273.

[41] Friedman M., Levy A., and Millstein T.: Navigational plans for data integration. In Proc. of the 16th Nat. Conf. on Artificial Intelligence (AAAI99), pages 67C 73. AAAI Press/The MIT Press, 1999.

[42] Guber T. R.: Toward principles for the design of ontologies used for knowledge sharing. In International Journal Human-Computer Studies, 43(5-6):907-928, 1995.

[43] Gómez-Pérez, A.: Knowledge sharing and reuse. The Handbook on Applied Expert Systems (1998).

[44] Gangemi A., Guarino N., Masolo C., and Oltramari A.: Sweetening wordnet with DOLCE. AI Magazine, 24(3):13C24, 2003.

[45] Grau C. B., Horrocks I., Kazakov Y., and Sattler U.: Just the right amount: extracting modules from ontologies. In Proceedings of the 16th International Conference on World Wide Web (WWW-2007), Banff, Alberta, Canada, May 8-12, 2007, pp. 717-726. ACM.

[46] Grau C. B., Horrocks I., Kazakov Y., and Sattler U.: Modular reuse of ontologies: Theory and practice. Journal of Artificial Intelligence Research, 31:273-318, 2008.

[47] Grosof N. B., Horrocks I., Volz R., Decker S.: Description logic programs: combining logic programs with description logic. In Proceedings of the Twelfth

International WorldWideWeb Conference, WWW2003, Budapest, Hungary, 20-24 May, 2003.

[48] Ghilardi S., Lutz C., and Wolter R.: Did I damage my ontology? A case for conservative extensions in description logics. In Proc. of KR-06, pages 187C197, 2006.

[49] Grau C. B., and Motik B.: Importing ontologies with hidden content. In Proceedings of the 22nd International Workshop on Description Logics (DL 2009). 2009

[50] Grau C. B., Motik B., Kazakov Y.: Import-by-query: Ontology reasoning under access limitations. In Proc. of the 21st Int. Joint Conf. on Artificial Intelligence (IJCAI 2009). AAAI Press. 2009. .

[51] Grau C. B., Parsia B., Sirin E., and Kalyanpur A.: Modularity and web ontologies. In Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning (KR-2006), Lake District of the United Kingdom, June 2-5, 2006, pp. 198-209. AAAI Press.

[52] Hull R.: Managing semantic heterogeneity in databases: A theoretical perspective. In Proc. of the 16th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS97), 1997.

[53] Haarslev V., Möller R.: RACER system description. In Proc. IJCAR 2001, Siena, Italy, 2001.

[54] Halevy Y. A.: Answering queries using views: A survey. Very Large Database Journal, 10(4):270C294, 2001.

[55] Horrocks I., Kutz O., and Sattler U.: The Even More Irresistible SROIQ. Proc. of the 10th International Conference of Knowledge Representation and Reasoning (KR-2006, Lake District UK), 2006.

[56] Van Harmelen F., Lifschitz V., Porter B. (eds): Handbook of knowledge representation Amsterdam ; Boston : Elsevier, 2008.

[57] Horrocks I., Li L., Turi D. and Bechhofer S.: The instance store: DL reasoning with large numbers of individuals. In Proc. of the 2004 Description Logic Workshop (DL 2004), pages 31-40, 2004.

[58] Haase P. and Motik B.: A mapping system for the integration of OWL-DL ontologies. In Axel Hahn, Sven Abels, Liane Haak, IHIS 05: Proceedings of the first international workshop on Interoperability of heterogeneous information systems pp. 9-16. ACM Press, November 2005.

[59] Horrocks I., Patel-Schneider F. P., Van Harmelen F.: From SHIQ and RDF to OWL: the making of a web ontology language. In Journal of Web Semantics, 1(1), 7-26, 2003.

[60] Horrocks I. and Sattler U.: Decidability of SHIQ with complex role inclusion axioms. Artificial Intelligence 160 (2004), 79C104.

[61] Hudek A. K. and Weddell G.: Binary absorption in tableaux-based reasoning for description logics. In Proc. DL 2006, Windermere, UK, 2006.

[62] Jiménez-Ruiz E., Cuenca Grau B., Horrocks I., and Berlanga R.: Building ontologies collaboratively using contentCVS. In Proc. of the 2009 Description Logic Workshop (DL 2009), 2009.

[63] Haase P., Wang Y.: A decentralized infrastructure for query answering over distributed ontologies. In Proceedings of The 22nd Annual ACM Symposium on Applied Computing (SAC). ACM Press, Seoul, Korea, March 2007.

[64] Klyne G., Carroll J.: RDF concepts and abstract syntax. http://www.w3.org/TR/ rdf-primer/, 2004.

[65] Klein M., Fensel D., Kiryakov A., and Ognyanov D.: Ontology versioning and change detection on the web. In: Proc. of the 13th European Conf. on Knowledge Engineering and Knowledge Management (EKAW02) (Siguenza, Spain). (2002) 197-212.

[66] Konev B., Lutz C., Walther D., and Wolter F.: Semantic modularity and module extraction in description logics. In Proceedings of ECAI-08, pages 55C59, 2008.

[67] Kirk T., Levy A. Y., Sagiv Y., and Srivastava D.: The Information Manifold. In Proceedings of the AAAI 1995 Spring Symp. on Information Gathering from Hetero- geneous, Distributed Enviroments, pages 85C91, 1995.

[68] Kalfoglou Y. and Schorlemmer M.: Ontology mapping: the state of the art. The Knowledge Engineering Review, 18(1):1C31, 2003.

[69] Kozaki K., Sunagawa E., Kitamura Y., Mizoguchi R.: Distributed and collaborative construction of ontologies using Hozo. In Proceedings of the Workshop on Social and Collaborative Construction of Structured Knowledge (CKC 2007). Banff, Canada.

[70] Kontchakov R., Wolter F., and Zakharyaschev M. Can you tell the difference between DL-Lite ontologies? In Proc. of KR-08, pages 285C295, 2008.

[71] Lederberg J. How Dendral was conceived and born. ACM Symposium on the History of Medical Informatics, 5 November 1987,

[72] Lenzerini M.: Data Integration: A Theoretical Perspective. In Proc. ACM Symposium on Principles of Database Systems (PODS 2002), pp. 233C246.

[73] Berners-Lee T., Hendler J., Lassila O: The semantic web. In Scientific American, 284(5):34C43, May 2001.

[74] Lutz C., Walther D. and Wolter F.: Conservative extensions in expressive description logics. In Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07), Hyderabad, India, January 2007, pp. 453-459. AAAI.

[75] Manolescu I., Florescu D., and Kossmann D.: Answering XML queries on heterogeneous data sources. In Proc. of the 27th Int. Conf. on Very Large Data Bases (VLDB 2001), pages 241C250, 2001.

[76] Motik B., Horrocks I. and Sattler U.: Bridging the gap between OWL and relational databases. J. of Web Semantics, 7(2):74-89, April 2009.

[77] Muñoz Garciá, O., Gómez-Pérez, A., Iglesias-Sucasas, M., Kim, S.: A workflow for the networked ontologies lifecycle: A case study in FAO of the UN. In Borrajo, D., Castillo, L., Corchado, J.M. (eds.) CAEPIA 2007. LNCS (LNAI), vol. 4788, pp. 200-209. Springer, Heidelberg.

[78] Mena E., Kashyap V., Illarramendi A. and Sheth, A.: Managing multiple information sources through ontologies: relationship between vocabulary heterogeneity and loss of information. In Knowledge Representation Meets Databases, Proceedings of the 3rd Workshop KRDB'96, Budapest, Hungary, Technical University of Aachen.

[79] Mayfield J., Labrou Y., Finin T.: Evaluating KQML as an agent communication language. In Intelligent Agents: II(eds Wooldridege M., Müller J. P. and Tambe M.), LNAI Volume 1037, pp.347-360. Springer, Berlin.

[80] Maedche A., Motik B., Silva N., and Volz R.: MAFRA - a mapping framework for distributed ontologies. In 13th European Conference on Knowledge Engineering and Knowledge Management EKAW, Madrid, Spain, 2002.

[81] Motik B., Sattler U.: A comparison of Rrasoning techniques for querying large description logic ABoxes. In M. Hermann and A. Voronkov, (eds), Proc. of the 13th International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR 2006), Vol 4246 of LNCS, Springer, 2006.

[82] Motik B., Shearer R., Horrocks I.: Optimized reasoning in description logics using hypertableaux. In Frank Pfenning, editor, Proc. of the 21st Conference on Automated Deduction (CADE-21), volume 4603 of LNAI, pages 67C83, Bremen, Germany, July 17−20 2007. Springer.

[83] Motik B., Shearer R., Horrocks I.: Hypertableau reasoning for description logics. Journal of Artificial Intelligence Research, 36:165C228, 2009.

[84] Motik B., Sattler U. and Studer R.: Query Answering for OWL-DL with Rules. Proc. of the 3rd International Semantic Web Conference (ISWC 2004), Hiroshima, Japan, November, 2004.

[85] N. F. Noy. Semantic integration: a survey Of ontology-based approaches. SIGMOD Record, Special Issue on Semantic Integration, 33, 4. Published in 2004.

[86] Noy N, Chugh A, Alani H: The CKC challenge: exploring tools for collaborative knowledge construction. In BMIR-2007. 2007: 1260.

[87] Noy F. N., Chugh A., Liu W., and Musen A. M.: A framework for ontology evolution in collaborative environments. In 5th ISWC, volume LNCS 4273, Athens, GA, 2006. Springer.

[88] Navathe S., Ceri S., Wiederhold G. and Dou J.: Vertical partitioning of algorithms for database design. ACM Trans. Database Syst. (December 1984), 9(4):680-710.

[89] Niles I. and Pease A.: Towards a standard upper ontology. In The 2nd International Conference on Formal Ontology in Information Systems (FOIS-2001), Ogunquit, Maine, 2001.

[90] Özsu M. T., Valduriez P.: Principles of distributed database systems. Prentice-Hall, 2nd edition, 1999.

[91] Pinto S., Gomez-Perez A., and Martins J.: Some issues on ontology integration. In Proceedings of the IJCAI-99 Workshop on Ontologies and Problem-Solving Methods(KRR5), Stockholm, Sweden, August 1999.

[92] Patel-Schneider P., Hayes P., Horrocks I: Web ontology language OWL Abstract Syntax and Semantics. W3C Recommendation.

[93] Russell S. and Norvig P.: Artificial intelligence: a modern approach. Prentice-Hall, 1995.

[94] Rahm E. and Bernstein P. A.: A survey of approaches to automatic schema matching. VLDB Journal, 10(4), 2001.

[95] Rector A., Schreiber G.: Qualified cardinality restrictions (QCRs): constraining the number of values of a particular type for a Property. W3C Working Draft (November 2 2005).

[96] Rector A., Welty C.: Simple part-whole relations in OWL ontologies. W3C Working Draft (August 11 2005).

[97] Sattler U.: Description Logics for Ontologies. In Proc. of the International Conference on Conceptual Structures (ICCS 2003), volume 2746 of Lecture Notes in Artificial Intelligence. Springer Verlag, 2003.

[98] Sunagawa E., Kozaki K., Kitamura Y., Mizoguchi R.: An environment for distributed ontology development based on dependency management. In Fensel, D., Sycara, K.P., Mylopoulos, J. (eds.) ISWC 2003. LNCS, vol. 2870. Springer, Heidelberg.

[99] Shearer R., Motik B., Horrocks I.: HermiT: A Highly-Efficient OWL Reasoner. In Alan Ruttenberg, Ulrile Sattler, and Cathy Dolbear (eds), Proc. of the 5th Int. Workshop on OWL: Experiences and Directions (OWLED 2008 EU), Karlsruhe, Germany, October 26C27, 2008.

[100] Sirin E., Parsia B., Grau C. B., Kalyanpur A., Katz Y.: Pellet: a practical OWL-DL reasoner. In Journal of Web Semantics 5 (2) (2007), 51C53.

[101] Seidenberg J., Rector A.: Web ontology segmentation: analysis, classification and use. In 15th International World Wide Web Conference, May 2006.

[102] Seidenberg J., Rector A.: The state of multi-user ontology engineering. In Proceedings of the 2nd International Workshop on Modular Ontologies, WoMO 2007, Whistler, Canada, October 28, 2007.

[103] Seidenberg J., Rector, A.L.: A methodology for asynchronous multi-user editing of semantic web ontologies. In Proc. of K-CAP. (2007) 127C134.

[104] Schmidt-Schauß M., Smolka G.: Attributive concept descriptions with complements. Artificial Intelligence, Elsevier, 48(1), 1-26, 1991.

[105] Staab S., Studer R.(eds): Handbook on ontologies. In International Handbooks on Information Systems. Springer, 2004.

[106] Sattler U., Schneider T., Zakharyaschev M.: Which Kind of Module Should I Extract? Description Logics 2009.

[107] Serafini L. and Tamilin A.: Local tableaux for reasoning in distributed description logics. In V. Haarslev and R. Moeller, editors, Proceedings of the 2004 Int. Workshop on Description Logics (DL2004), pages 100-109, CEUR-WS, 2004.

[108] Sacca D. and Wiederhold G.: Database partitioning in a cluster of processors. ACM Trans. Database Syst. (October 1985), 10(1): 29-56.

[109] Tsarkov D., Horrocks I.: Efficient reasoning with range and domain constrains. In Proc. DL 2004, whistle, BC, Canada, 2004.

[110] Tsarkov D., Horrocks I.: FaCT++ description logic reasoner: system description. In Proc. IJCAR 2006, Seattle, WA, USA, 2006.

[111] Tudorache T., Noy F. N., Tu S., Musen A. M.: Supporting collaborative ontology development in protégé. In Musen 7th International Semantic Web Conference (ISWC 2008), Karlsruhe, Germany, Springer.

[112] Thomas e., Pan J. and Ren Y.: TrOWL: Tractable OWL 2 reasoning infrastructure. In Proc. of the Extended Semantic Web Conference (ESWC2010).

[113] Tempich C., Pinto S., Sure Y. and Staab S.: An Argumentation Ontology for DIstributed, Loosely-controlled and evolvInG Engineering processes of oNTologies (DILIGENT). In Proc. of ESWC2005, Greece, pp. 241-256, 2005.

[114] Valo A., Hyvonen E. and Komurainen V.: A tool for collaborative ontology development for the semantic web. In Proc. of DC 2005, Madrid, Spain, 2005.

[115] N. Villanueva-Rosales and M. Dumontier. YOWL: an Ontology-Driven Knowledge Base for Yeast Biologists. *Journal of Biomedical Informatics*. Available online May 11, 2008.

[116] Wooldridge, M.: An introduction to multiagent systems. John Wiley and Sons, Chichester (2009).

[117] Wooldridge M. and Jennings N. R.: Intelligent agents: theory and practice. In Knowledge Engineering Review, 10(2):115-152, 1995.