A META-LOGICAL FRAMEWORK FOR AGENT COMMUNICATION OF SEMANTIC WEB INFORMATION

VUONG TRAN XUAN

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENT FOR THE DEGREE OF
MASTER OF ENGINEERING IN COMPUTER ENGINEERING
SCHOOL OF GRADUATE STUDIES
KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG
2005

ISBN 974-15-2197-9

หัวข้อวิทยานิพนธ์	A Meta-logical Framework for Agent Communication of Semantic Web Information
ผู้วิจัย	Vuong Tran Xuan
รหัสนักศึกษา	47060834
ระดับการศึกษา	วิศวกรรมศาสตรมหาบัณฑิต
ภาควิชา	วิศวกรรมคอมพิวเตอร์
ปี	2005
อาจารย์ผู้ควบคุมวิทยานิพนธ์	Asst. Prof. Dr. Visit Hirankitti
อาจารย์ผู้ควบคุมวิทยานิพนธ์ร่วม	Prof. Dr. Hidekazu Tsuji

บทคัดย่อ

Semantic Web ถูกพัฒนาขึ้นเพื่อแลกเปลี่ยนข้อมูลเว็บที่สื่อความหมายด้วย metadata ระหว่างเครื่องคอมพิวเตอร์ด้วยกันเพื่อให้สามารถประมวลผลได้อย่างอัตโนมัติ มีการคาดหมายว่า Semantic Web จะเข้ามาแทนที่เว็บปัจจุบันในเวลากันใกล้ ในการแทนความหมายเนื้อหาบนเว็บด้วย metadata ร่วมกับ ontology สำหรับโดเมนนั้นๆ ทำให้เว็บสามารถให้บริการที่มีความสามารถสูงขึ้น ส่วนความสำเร็จของ Semantic Web นั่นจะขึ้นอยู่กับความง่ายและการที่มีรูปแบบเดียวกันในการติดต่อสื่อสาร การคิดหาเหตุผลที่เกี่ยวกับข้อมูลเว็บที่สื่อความหมายได้และการแลกเปลี่ยนข้อมูลเหล่านี้ระหว่างเครื่องคอมพิวเตอร์ ในวิทยานิพนธ์ฉบับนี้ได้มีการพัฒนาเฟรมเวิร์คสำหรับการสื่อสารข้อมูลเนื้อหาของ Semantic Web โดย agent การแทนรูปแบบของ ontology การสร้าง agent และการสื่อสารระหว่าง agent จะอยู่ในรูปแบบของ meta-logic ทั้งหมด agent ตัวเดี่ยวๆ ที่ทำหน้าที่เป็น meta-logical system เรียกใช้เพรดิเคท demo() เพื่อทำงานเป็นเครื่องจักรวินิจฉัย (inference engine) และใช้ meta-program ที่แปลงมาจาก ontology บน Semantic Web ร่วมกับ axioms เพื่อเป็น assumption ของระบบ agent เหล่านี้สามารถคิดหาเหตุผลโดยใช้ assumption ของตนเองรวมทั้งใช้ assumption ของ agent อื่นๆ ด้วยความสามารถดังกล่าว เมื่อ agent หลายๆ ตัวถูกสร้างขึ้นมาโดยเฟรมเวิร์คนี้ ชุมชน agent เหล่านี้สามารถติดต่อสื่อสารข้อมูลเนื้อหา Semantic Web ระหว่างกันได้บนอินเตอร์เน็ต

คุณค่าที่เกิดจากงานวิจัยในวิทยานิพนธ์ฉบับนี้คือ การนำเสนอวิธีการแบบเดียว โดยใช้เพียง meta-logic สำหรับการแทนรูปแบบ ontology การคิดหาเหตุผลแบบ meta-reasoning การคิดหาเหตุผลแบบกระจาย การสื่อสารระหว่าง agent ด้วยวิธีการดังกล่าวทำให้เฟรมเวิร์คที่นำเสนอสามารถคิดหาเหตุผลกับ multiple distributed ontologies บน Semantic Web ได้ รวมทั้งสามารถรวมเอาการสื่อสารระหว่าง agent หลายตัวเข้าไว้ในเฟรมเวิร์กเดียวกัน

Thesis Title	A Meta-logical Framework for Agent Communication of Semantic Web Information
Student	Vuong Tran Xuan
Student ID.	47060834
Degree	Master of Engineering
Programme	Computer Engineering
Year	2005
Thesis Advisor	Asst. Prof. Dr. Visit Hirankitti
Thesis Co-Advisor	Prof. Dr. Hidekazu Tsuji

ABSTRACT

Semantic Web, envisioned as the next generation of the current Web, is aimed at providing an automated information access based on machine-processable semantic metadata of web resources. The explicit representation of such semantic metadata, accompanied with domain ontologies, will enable a Web that facilitates a qualitatively new level of services. The success of the Semantic Web would be determined by how easy and uniform to access to, reason with, and exchange of semantic information among computers. In this thesis we develop a framework for agent communication of Semantic Web information. The representation of Semantic Web ontologies, the agent, and the communication between agents are characterized in meta-logic. One single agent, understood as a meta-logical system, adopts a demo(.) predicate as its inference engine and meta-programs, transformed from some Semantic Web ontologies as well as Semantic Web axioms, as its assumptions. Such an agent can reason with its assumptions as well as other agents' assumptions. With this capability, when several agents are created using our framework, the community of these agents can uniformly communicate Semantic Web information between each other on the Internet.

The contribution of this thesis is to provide a single approach, for ontology representation, meta-reasoning, distributed reasoning, and agent communication in a uniform manner based on meta-logic. With this approach, it allows the framework to be able to reason with multiple distributed ontologies on the Semantic Web and able to embrace communication of multi-agents into one single framework.

Acknowledgements

This thesis is the result of two years of work whereby I have been accompanied and supported by many people. It is a pleasant time that I would like to express my sincere gratitude and appreciation to all of them.

The first person I would like to give a special thank is my direct supervisor, Asst. Prof. Dr. Visit Hirankitti, Department of Computer Engineering, King Mongkut's Institute of Technology Ladkrabang. During two years I have known him as a sympathetic and principle-centered person. His overly enthusiasm, meticulous and integral view on research as well as his mission for creating only high quality works, have made a deep impression on me. I owe him lots of gratitude for his encouragement, guidance, patience, and kindness. During my study, I have been influenced by his logical view and scientific methodology on research.

I am very grateful to my supervisor, Prof. Dr. Hidekazu Tsuji, Head of the Department of Media Technology, Tokai University, for his constructive discussions, and for his important support throughout this work.

I would like to express my gratitude to people and friends in the Department of Computer Engineering, especially to lecturers for their lectures and encouragement.

My warm and loving appreciation is due to my family, my friends, and my colleagues for their continuous support, understanding, and encouragement. I have been always full of energy for my study in their warm arms.

The financial support of this study for a high quality working environment was generously funded by the Japan International Cooperation Agency (JICA) under the ASEAN University Network/ Southeast Asia Engineering Education Development Network (or AUN/SEED-Net) Project. I would also like to warmly thank the secretariats of the AUN/SEED-Net project who have always given helpful and kind support.

My gratitude would be incomplete if I would forget to thank people in the Faculty of Information Technology, Ho Chi Minh City University of Technology for their kind and essential help, in particular Dr. Quan Thanh Tho, Mr. Tran Quang, Assoc. Prof. Dr. Cao Hoang Tru, and Ms. Tran Thi Phuc, to name a few.

Finally, I would also like to thank the members of my thesis committee who monitored my work and took effort in reading and providing me with valuable comments and suggestions on earlier versions of this thesis.

Vuong Tran Xuan

January 10, 2006

Bangkok, Thailand

Contents

page

Contents (cont.)

page

Contents (cont.)

Contents (cont.)

page

Contents (cont.)

page

List of Tables

List of Figures

List of Figures (cont.)

Chapter 1

Introduction

1.1 The Problem

Over the last decade the World Wide Web (or briefly "WWW") has gained astonished success. The exponential growth in the number of web pages makes it become an information resource with virtually unlimited potential. However its potential has not opened up to the full extent because of the limitations of the current web technologies which are almost suitable for human interpretation of information. This leads to an effort of developing the next generation of the WWW which is called "Semantic Web" (or briefly "SW"). The SW is aimed at making web resources more accessible and processible to automated processes by adding semantic annotations—metadata that is used to describe web contents. In the SW ontologies play a crucial role in enabling web-based knowledge processing, sharing, and reuse between applications. Generally defined as shared formal conceptualizations of particular domains, ontologies provide a common understanding of those domains that can be used to facilitate communication between people and applications. Similarly to the current Web where web contents are distributed and linked web pages, the SW would contain multiple distributed but linked ontologies. Different companies, organizations, institutions, etc. would produce their information contained in those ontologies. A vast and growing amount of ontologies will be available on the Web and these ontologies can be then processed by intelligent software tools. Integration of different information sources defined in multiple distributed ontologies leads to the following potential problems that need to be concerned.

- **Ontology representation**

Some SW languages like Resource Description Framework (or briefly "RDF") [Lassila & Swick, 1999], RDF Schema [Brickley & Guha, 2004], and Web Ontology Language (or briefly "OWL") [Miller & Hendler, 2004] have a well-defined syntax and semantics and ontologies encoded in these languages are suitable for interchange

between applications on the Web. However we need to map an ontology from its original form, e.g. defined in XML-based syntax of RDF, RDF Schema, and OWL, to a known logical formalism and to use some reasoning mechanisms for that formalism to be able to process information of the ontology.

- **Ontology processing**

Basing on information defined in ontologies we can exploit explicit as well as implicit knowledge from these ontologies. This means that we can describe explicitly information of some parts of a domain in some ontologies. Given these ontologies and inference rules we then can derive implicit knowledge by using some inference mechanisms. Therefore in order to process ontologies, not just querying functions but also reasoning mechanisms must be applied for software systems.

- **Multiple ontologies**

It is not always possible to create an ontology which includes all necessary information. Instead that information is usually scattered via various ontologies. An application must have ability of concerning with multiple ontologies. There are many problems related to multiple ontologies such as ontology aligning, versioning, mapping, etc. which have been researched in many literatures. Regard to the multiple ontologies problem, this thesis concentrates on reasoning with multiple ontologies. That is how to extract, integrate, and reason with information from different sources of ontologies.

- **Distributed ontologies**

As in the current Web where we often need to combine information from several web pages that are distributed over the Internet, in the SW applications may also have to work with distributed ontologies. An application may retrieve necessary ontologies from the Web and then process them locally. This approach is simple and easy to be implemented but may be not applicable in many cases where large-scale sources of information are needed. A promising alterative is to build reasoning services working on each source of ontologies and these services can co-operate each other in a distributed reasoning manner in order to process multiple distributed ontologies.

We focus on above problems and in this thesis we propose a meta-logical framework for agent communication of SW information (described in SW ontologies). The framework can work with multiple distributed ontologies in an agent-based manner.

1.2 The Motivation

We realize that there are relatively close relations among three fields: Semantic Web, logic programming, and agent technology. These fields motivated the existence of this thesis. And the research work on this thesis suggests that logic programming and agent technology are suitable and applicable to develop applications which are used for processing SW information in the SW.

- Web ontology languages and logic

Ontology has been an essential part of the SW as it is used to describe a domain of discourse. SW resources are described by ontology-based metadata formulated in several XML-based markup languages. For instance, RDF provides a basic data modeling in the form of subject-predicate-object triples to make sentences describing web resources, and RDFS, OWL, DARPA Agent Markup Language and Ontology Inference Language (or briefly "DAML+OIL") [Horrocks, 2002] are used to define classes, instances, properties, and their relationships for those resources. The development of these languages has been influenced by different concepts such as semantic structures and frame representations, object-oriented approaches, Description Logic (or briefly "DL"), and first-order logic (or briefly "FOL").

RDF can be seen as a language of first order predicate logic, with resources being subjects, properties being predicates and property values either being subjects or constants [Peer, 2002]. In addition, a graph representation of RDF data model can be understood as a semantic network (or briefly "SN") [Shukla, 2003]. SN is a knowledge representation schema involving nodes and links (arcs) between nodes. The nodes represent objects or concepts and the links represent relations between nodes. In RDF graph model, each RDF triple is represented as a node-arc-node link in which the nodes are subjects and objects and the arcs are properties.

A particular use of semantic networks is to represent logical descriptions. Conceptual Graph (or briefly "CG") is a kind of SN related to the Existential Graphs of Peirce. The aim of using CG is to describe concepts and their relations in a way that is both close to human language and formal, in order to provide a readable but formal design and specification language. Basically, a CG is a bipartite graph containing two kinds of nodes, called concepts and conceptual relations, respectively. Arcs link concepts to conceptual relations, and each arc is said to belong to a conceptual relation. Therefore the RDF graph model also relates to CG concept [Corby et al., 2000].

An object-oriented approach also has an influence on the design of OIL [Fensel et al., 2001], a part of the DAML+OIL language from which OWL was derived. In addition, both DAML+OIL and OWL are developed based on DL, especially OWL DL, a sub-language of OWL, is designed to support DL which is a subset of the FOL.

- **SW and logic programming**

SW languages and logic programming languages are similar in that they are both declarative. In logic programming, a domain of discourse is described in terms of objects and their relationships together with rules for deduction. In the SW, just some parts of information of resources are explicitly described. Given some ontologies and rules, implicit knowledge can be derived by using an inference engine. Hence, a logical system should be used for this purpose. Furthermore, as mentioned above, ontology languages like DAML+OIL and OWL are heavily influenced by DL. To exploit this feature of SW languages a mapping—with some restrictions—from DL ontology to a Horn-clause program was proposed in [Grosof & Horrocks, 2003].

- **SW and agent technology**

The SW provides an environment where information is structured in an explicit machine-processable way and is distributed over the Internet. The goal is to facilitate resources discovery, intelligent searching, information integration, electronic commerce, etc. With such environment we need an efficient paradigm for developing intelligent software tools which are capable of processing SW information automatically.

Agent technology has been investigated for years and has gained significant success in many fields like grid computing, artificial intelligence, scheduling and planning, online and parallel algorithms, etc. This technology can be efficiently used for information processing systems that are distributed, large, open, and heterogeneous.

Therefore, agent technology is a promising key approach in realizing the SW vision, enabling more flexible and independent interaction between applications. The SW can be seen as a cyberspace for software agents of different kinds, goals, and architectures. Such agents must efficiently realize the goals delegated to them by their users. In order to achieve that they must perceive the external environment, interact with other agents and users. In an open environment like the SW it is often impossible for a single agent performing efficiently its tasks without co-operating with other agents. Sophisticated agent interaction mechanisms and services are needed. It seems that the process automation needed in the SW makes the development of agent systems feasible and promising.

Because of their promising features for the SW, unify SW technology, logic programming, and agent technology into a single framework. In order to do that, in this thesis we apply meta-logic to characterize and develop our framework.

1.3 The Objectives

The overall purpose of this thesis is to develop a meta-logical framework for agent communication of SW information that unifies SW technology, logic programming, and agent technology. In more details, we summarize concrete objectives as follows.

- To find an efficient method for representation of SW ontologies which distinguishes various meta-levels of knowledge represented in ontologies.
- To develop a meta-interpreter as inference engine for SW information processing system. Such meta-interpreter can reason with its ontologies as well as with others' ontologies and communicate with external systems.
- To characterize and develop a framework for SW agents.
- To facilitate communication of SW information among SW agents.
- To reason with multiple distributed ontologies.

- To develop and demonstrate a software system which is implemented for our meta-logical framework.

1.4 The Assumption and Scope

Firstly our framework supports ontology languages like RDF, RDF Schema, OWL, and Semantic Web Rule Language (or briefly "SWRL") [Horrocks et al., 2004].

Secondly, there are various problems concerning with multiple ontologies like ontology aligning, ontology versioning, etc. Different organizations create domain-specific ontologies capturing specific aspects of their knowledge. Special mapping ontologies are created to link different concepts used in those ontologies, creating bridges between separated pieces of knowledge. These bridges are then used to perform cross ontology information search and retrieval from the ontologies. Various techniques are developed to cope with these problems which will be reviewed in chapter 2. In this thesis we concentrate on the problem of reasoning with multiple distributed ontologies rather than creation, maintenance or mapping of ontologies. With this assumption the system will not focus on consistency checking, version checking which should be done in the process of creating or maintaining ontologies instead. Exploitation and combination of information from multiple distributed ontologies is the main task of our agent-based system. Various agents will co-operate and exchange their information reasoned from multiple ontologies to perform user's tasks or queries.

Thirdly, to be able to build a distributed reasoning system, besides the ability of reasoning with multiple ontologies we also focus on agent communication so that each agent can work autonomously as well as communicate with other agents for exchanging their information. An agent in our system has important characteristics for that purpose like autonomy, reactivity, pro-activity, social ability. Some other properties like mobility have not been exploited. Besides the agent architecture is partly based on the Beliefs, Desires, and Intentions (or briefly "BDI") model. For agent communication, Knowledge Query and Manipulation Language (or briefly "KQML") is used as a communication language while OWL and Prolog are used as content languages.

Finally, fundamentally, in this thesis we regard SW ontology representation, ontology reasoning, and agent communication as meta-logical formalism.

1.5 The Research Methodology

- **Investigating methods for knowledge representation of ontologies**

There are relations between SW languages like RDF, RDF Schema, OWL and knowledge representation. We analyze RDF data model, RDF Schema, and OWL languages in order to find a way of representing information defined in an SW ontology in terms of logical statements. Such logical representation must have a formal syntax and semantics as well as facilitate reasoning ability of our system.

- **Specifying language elements of an SW ontology**

Basic elements of an ontology are considered so that we can define and separate ontology information into different meta-levels. Basing on this analysis meta-languages are designed to express information at corresponding meta-levels. Vocabulary, syntax, and semantics of these meta-languages will be clearly defined.

- **Working with multiple distributed ontologies**

A study on existing approaches in the knowledge representation field for representing and reasoning with multiple theories is performed. Similarly to the knowledge representation field, in our system ontologies are treated as different theories and pieces of information in an ontology will be stored along with their ontology name. This helps a reasoning system know which sources of ontologies are used to infer a conclusion. Ontology reasoning axioms are also considered to build a common knowledge that can be applied for reasoning systems working with SW ontologies.

- **Proposing an approach for semantic integration**

To integrate information defined in various ontologies we must define connections between them. Basing on these links a reasoning system can extract and combine information over various sources of information.

- **Developing a meta-interpreter to reason with SW ontologies**

Various problems related to reasoning with multiple ontologies are considered. The meta-interpreter must have ability of reasoning with its ontologies, extracting and

combining information from various sources, retrieving SW ontologies from the Internet as well as communicating information with each other.

- **Developing agent communication of SW information**

Agent techniques such as agent communication language, content language, interaction protocols, etc. will be considered. They are adapted and integrated into our meta-interpreter, which can be seen as an agent's inference engine, so that an agent developed by our framework can cooperate and exchange their information on the Web.

1.6 The Contributions

In this section we first present general contributions are made in this thesis. We then summarize the content of each chapter; therefore provide an overall view of the whole thesis.

1.6.1 General Contributions

- The main contribution of this thesis is to provide a single approach, for ontology representation, meta-reasoning, distributed reasoning, and agent communication in a uniform manner based on meta-logic.
- Various meta-languages based on meta-logic are designed to represent SW ontologies in terms of logical statements which enable the development of an efficient reasoning mechanism.
- A meta-interpreter is developed to reason and exchange SW information defined in SW ontologies.
- Our meta-logical system is capable of working and reasoning with multiple distributed ontologies.
- SW agents are characterized and developed in order to utilize and communicate SW information in a multi-agent framework.

1.6.2 Thesis Overview

- **Chapter 2 Research Works in Semantic Web**

This chapter first introduces the motivation and the vision of the SW, discusses concepts and technologies related to the SW like logic, logic programming, ontology, and agent technology. These concepts and technologies form the foundation of the approach in this thesis. The chapter then briefly surveys previous works concerned with various approaches for processing SW information, with approaches related to logic programming and agent technology. It suggests the need of a unified meta-logical framework based on meta-logic and agent technology for manipulating SW information.

- **Chapter 3 A Meta-logical Framework for Agent Communication of Semantic Web Information**

This chapter presents the overview of our meta-logical framework based on agent communication. The roles of various agents used in the system are briefly explained. The chapter then introduces meta-languages defined for expressing SW ontologies in terms of meta-programs which can be seen as an agent's knowledge base. The meta-interpreter is described as an agent's inference engine. The chapter also explains how various agents in the system can co-operate and communicate each other in order to form the agent communication of SW information.

- **Chapter 4 Meta-languages for Semantic Web Ontologies**

This chapter first analyzes language elements and meta-information of SW ontologies. Basing on this analysis we describe the design of various meta-languages that are built for expressing information of SW ontologies at different meta-levels. They include a meta-language for object information, a meta-language for meta-information, and a meta-language for meta-information related to multiple SW ontologies. The vocabulary, syntax, and semantics of these meta-languages are clearly discussed and defined.

- **Chapter 5 Meta-programs of Semantic Web Ontology**

This chapter shows how to express and represent information defined in SW ontologies in terms of meta-programs by using our meta-languages described in the

chapter 4. Logical representation of object information forms the meta-program for object information while logical representation of meta-information forms the meta-program for meta-information. Another meta-program containing inference axioms that are required for reasoning with SW ontologies is also presented.

- **Chapter 6 Communicative Semantic Web Meta-interpreter**

This chapter concerns with the design and development of a meta-interpreter which can reason with meta-programs transformed from SW ontologies, retrieve SW ontologies from the Internet, and communicate with other systems. The meta-interpreter is developed by adapting and extending the Vanilla interpreter, a well-known interpreter in logic programming field, is used to manipulate object programs.

- **Chapter 7 Agent Communication of Semantic Web Information**

This chapter first describes a single SW agent architecture with agent characteristics, architectural components, and agent life cycle. After that the chapter will discuss various aspects related to the development of agent communication of SW information, including agent location, agent interface, agent communication language, content language, interaction protocols, and agent roles.

- **Chapter 8 The MAC-SWI Framework Compared to Other Related Works**

This chapter gives comparison between the work of the thesis and other related works such as those concerned with logic programming, agent technology, multiple ontologies, and distributed reasoning for processing SW information.

- **Chapter 9 Conclusion**

This chapter presents main results achieved in the thesis, discusses some close related fields like SW services and application domains for information retrieval, digital library, distributed scheduling, electronic commerce, etc. that can benefit our framework. The chapter also suggests what might be further study to extend the work in this thesis to address issues like proof, trust, security, etc.

Chapter 2

Research Works in Semantic Web

In this chapter we will present a brief introduction of related fields in Semantic Web, like the WWW, ontology, logic, logic programming, and agent technology. These concepts form the basic background for the research in this thesis. We next consider previous works which are related to our research before we introduce our proposal.

2.1 Semantic Web Concepts

2.1.1 Introduction

The invention of the WWW has significantly changed the way human exchanging information around the world. The WWW has an impressive success, in terms of both its available information and the continuing rapid growth of its human users. This makes the WWW become an important part of our everyday life. Companies, organizations, and people use the WWW as a powerful medium to exploit and share their information. Companies may advertise and sell their products on websites. Educational institutes offer teaching materials and online training services. Anyone can search, exchange information published on the Web. The success of the WWW is mainly based on its simplicity which provides software developers, information providers, and users an easy way to create and to access to new media. Unfortunately, the popularity of the WWW leads to an exponential growth in the number of web pages. This makes it difficult for human users finding and accessing the desired information. To cope with the enormous quantity of data, we need to hand off portions of these tasks to computers. In order to help softwares understand and process automatically information for assisting users complete necessary tasks, information must be given in a well-defined meaning format. This forms the vision of the Semantic Web, the next generation of the WWW. In the SW the meaning of information on a web page is formalized by using semantic metadata which is based on concepts defined in ontologies. In this thesis we consider that an ontology includes both concepts of a domain and information describing its individuals. Therefore similarly to the current web where web pages are distributed and linked

together, the content of the SW would be distributed but linked ontologies. This is a meaningful space where software agents can be used to autonomously perform desired tasks on behalf of human users. Such agents can scour the Internet to find necessary information that can answer the questions posed by their owners. Besides that to obtain the SW's vision two major tasks needed to be studied are representation and interpretation of web contents. Both these works have been influenced by various concepts including notations of various logics.

In the following sections we will consider the motivation behind the move from the WWW to the SW, the definition of SW architecture, important concepts which support for the development of the approach in this thesis such as ontology, logic, logic programming, and agent technology.

2.1.2 World Wide Web towards Semantic Web

In the current Web most of its contents are suitable for human consumption involving activities like seeking and making use of information, receiving catalogs of online stores, filling out forms for ordering products, etc. Such activities are tedious and should be supported by computers and various software technologies have been developed to assist human users to collect necessary information from huge sources on the Internet. At present finding information on web pages is heavily based on syntax- and text-oriented search. This means that tools like search engines often find headers, keywords, and other tags within a document to specify relevant information and return results to a user. The user then needs to manually find and select his desired information in those search results which often contain a huge amount of irrelevant documents and may miss the relevant materials.

The current Web is mainly based on the use of Hypertext Markup Language (or briefly "HTML") to encode, exchange, and display web contents over the Internet. And HTML documents provide software tools, mainly Web browsers, with a set of instructions to render and present their information.

HTML was simply designed for representation of web page contents so the semantics of these contents is not accessible for software tools. On the one hand, HTML's simplicity has helped spur the Web's fast growth. On the other, its simplicity

seriously has hampered more advanced web applications in many domains and for many tasks. This led to the creation of Extensible Markup Language (or briefly "XML") which allows users define arbitrary domain and task-specific extensions for information contents. However XML and related techniques provide just a surface syntax for structured documents but impose no semantic constraints on the meaning of contents.

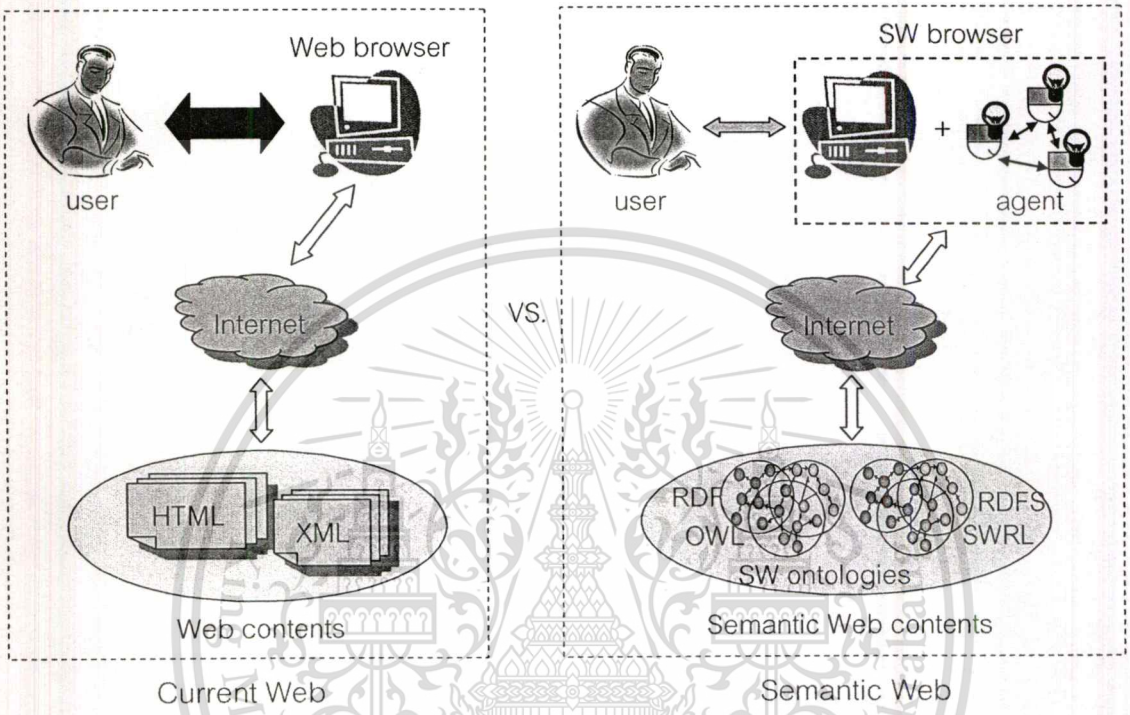


Figure.2.1 World Wide Web towards Semantic Web

The rapid growth in both number of users and amount of information as well as the limitations of current web technologies motivate the development of the SW which has evolved from the WWW. In general the aim of the SW is to allow much more advanced knowledge based applications that can benefit SW technologies. More concretely, in the Semantic Web,

- Information will be organized in conceptual spaces according to its meaning.
- Automated tools will be developed to support creating, maintaining, and extracting explicit and implicit information.
- Traditional keyword-based search will be replaced by query answering mechanisms and desired knowledge will be retrieved, extracted, and presented in a human-friendly way.
- Information is collected and combined over several sources of documents.

To be able to obtain the vision of the SW in which information can be shared and reused across application, enterprise, and community boundaries, the semantics of information must be accessible to computers. Figure.2.1 illustrates the vision of the SW where we can build software agents that capable of automatically working with SW ontologies in order to reduce human's effort for interpreting web contents.

In the next section we will consider the infrastructure of SW technology for embedding semantics of information on web pages by annotating web contents with semantic metadata.

2.1.3 Semantic Web Architecture

The development of the SW proceeds in steps, each step building a layer on top of another. Basically, there are seven layers on the architecture of the SW, the upper layer will use and extend the lower one. Figure.2.2 depicts this architecture and we will explain the role as well as the relationships between these layers in the following parts.

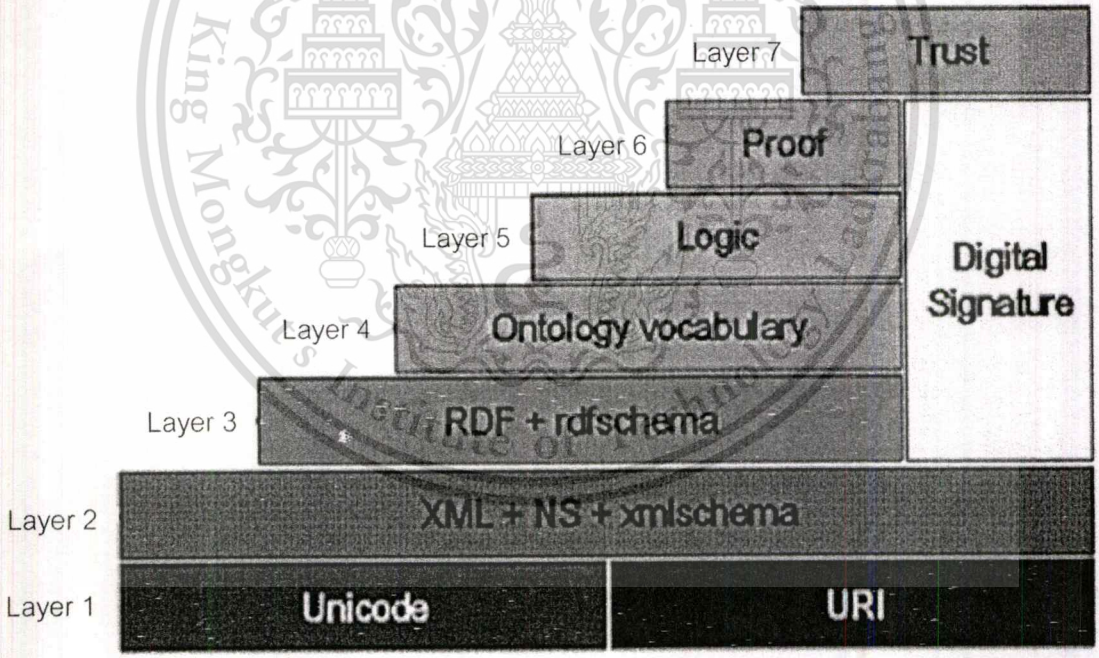


Figure.2.2 Layers of the Semantic Web architecture

- **Layer 1: Unicode and URI (URL, URN)**

Here, URI stands for Uniform Resource Indicator, URL stands for Uniform Resource Locator, and URN stands for Uniform Resource Name. We can say that URL and URN are subsets of URI because a URI indicates anything in the world as a

resource, not just in the Internet. However, a URL indicates a resource that can be accessed from the Internet while a URN indicates a name of a resource that must remain unique even though the resource is not still available.

Unicode is a universal code that is a set of characters of all major languages in the world. It is introduced three formats for encoding Unicode characters; they are UTF-8 (1 byte), UTF-16 (2 bytes) and UTF-32 (4 bytes).

- Layer 2: XML, Namespaces and XML Schema

XML is a subset of the Standard Generalization Mark-up Language (or briefly "SGML"). A markup language is a language which is used for adding more information to an existing document; this information is often separated from the original one. Applications will use this markup information to process data. A web service may use XML tags to process messages encoded in XML and passed between applications.

For example, to express that Tim Berners-Lee is the creator of the website http://www.rdf.org we can encode this information in XML as follows.

```
<website>
  <url>
    http://www.rdf.org
  </url>
  <creator>
    Tim Berners-Lee
  </creator>
</website>
```

Figure.2.3 An XML example

Or we can encode above sentence in another way as following:

```
<website url="http://www.rdf.org"
      creator="Tim Berners-Lee">
</website>
```

A namespace is a reference to a definition by XML that is unique and universal. Namespaces are used to resolve the problem of two definitions having the same name.

XML Schema offers a significantly richer language for defining the structure of XML documents and its syntax is based on XML itself.

- **Layer 3: RDF and RDF Schema**

RDF is a language and actually is an application of XML. It is used to describe information of web resources. There are three syntaxes for RDF language: XML-syntax, Notation 3 and N-triples. Any resource can be described by RDF statements in terms of subject - predicate – object triples.

With the previous example, we can describe the sentence with an RDF graph model as in illustrated in Figure.2.4.

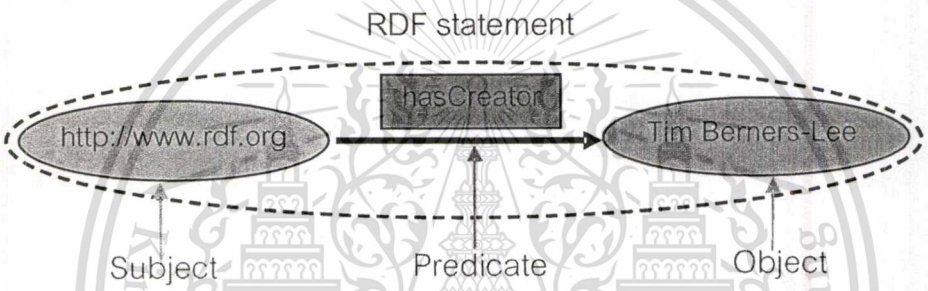


Figure.2.4 Graphic data model of an RDF triple

Or it can be expressed by XML-based syntax in Figure.2.5.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/DC/">
  <rdf:Description about="http://www.rdf.org/">
    <dc:Creator>
      Tim Berners-Lee
    </dc:Creator>
  </rdf:Description>
</rdf:RDF>
```

Figure.2.5 An RDF example

Here we use two namespace, the first `xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"` refers to the document described by RDF syntax

and the second `xmlns:dc="http://purl.org/DC/"` refers to a basic ontology about authors and publications of Dublin Core.

RDF is used for just the main purpose of describing resources. However RDF does not provide any assumption about the relationships between these resources. Therefore RDF Schema will provide primitives for expressing the information about semantics of concepts and their relationships in a domain.

For example, to describe information and relationship between Man and Human in that Human is a Class and Man is a sub class of Human, we can use RDF Schema primitives `rdfs:Class` and `rdfs:subClassOf` as following:

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://http://www.w3.org/2000/01/rdf-schema#">
  <rdfs:Class rdf:ID="Human">
    <rdfs:comment>This class for human</rdfs:comment>
  </rdfs:Class>
  <rdfs:Class rdf:ID="Man">
    <rdfs:comment>This class for men</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Human"/>
  </rdfs:Class>
</rdf:RDF>
```

Figure.2.6 An RDF Schema example

- Layer 4: Ontology

An ontology defines a common vocabulary for sharing and reuse information in a domain. It includes machine-interpretable definitions of concepts in the domain and relationships among them.

Some main purposes of creating an ontology are:

- Sharing a common understanding of the structure of information among people and/or software agents.
- Enabling reuse of domain knowledge.
- Making domain assumptions explicitly.
- Analyzing domain knowledge, etc.

RDF Schema provides just some basic notations for defining ontologies. OWL is a web ontology language which includes three subsets OWL Lite, OWL DL, and OWL Full. The latter bases on the former and provides more powerful expression than the former. More details about ontology are given in section 2.1.4.

- Layer 5: Logic

The logic layer allows defining rules that enable inferences, e.g. rules are defined to choose a course of actions and to answer questions.

- Layer 6, 7: Proof & Trust

The proof layer is required to provide explanations about the answers given by automated agents that consume the provided information. This requires the translation of an agent's internal reasoning process into a unifying proof representation language. Proof and trust mechanisms are still under research and development.

2.1.4 Semantic Web and Ontologies

Researchers in artificial intelligence (or briefly "AI") field first developed ontologies to facilitate knowledge sharing and reuse. Since the beginning of the 1990s, ontologies have become a popular research topic, and several AI research communities—including knowledge engineering, natural language processing, and knowledge representation—have investigated them. More recently, the notion of ontology is becoming widespread in fields such as intelligent information integration, cooperative information systems, information retrieval, electronic commerce, and knowledge management. Ontologies are becoming popular largely because of what they promise: a shared and common understanding that reaches across people and application systems. Many definitions of ontologies have surfaced in the last decade, but the one that is best to characterize ontology's essence is this: "An ontology is a formal, explicit specification of a shared conceptualization". In this definition, *conceptualization* refers to an abstract model of some phenomenon in the world that identifies relevant concepts of that phenomenon. *Explicit* means that the type of concepts used and the constraints on their use are explicitly defined. *Formal* means that

the ontology should be machine-understandable. Different degrees of formality are possible. And *shared* reflects the notion that an ontology captures consensual knowledge—that is, it is not restricted to some individuals but is accepted by a group.

In the SW ontology plays a crucial role in enabling web-based knowledge processing, sharing, and reuse across various applications. Ontologies serve as metadata schemas, providing a controlled vocabulary of concepts, each with explicitly defined and machine-processable semantics. By defining shared and common domain theories, ontologies help people and computers to communicate concisely—supporting semantic exchange, not just syntax interoperability.

Given an ontology of a domain, information about resources of that domain is linked to concepts and relationships in the ontology. With defined rules, software tools can be developed to gather, process, and exchange information from web resources. Besides that links between ontologies enable knowledge reuse in the sense that an ontology needs not redefine a concept existed in another. Multiple distributed ontologies represent a distributed knowledge base on the Web and they promote common understanding of concepts and resolve ambiguity resulting from similar terminologies.

We now consider some typical examples of using ontologies in the SW. First ontologies can be used in e-commerce to enable machine-based communication between buyers and sellers, or enable description reuse between different marketplaces. Search engines can also use ontologies to support finding web pages with some words that may be syntactically different but semantically similar.

2.1.5 Semantic Web and Logic

In this section we analyze some typical languages that are based on logic for expressing and specifying ontologies. Basing on such analysis we will realize the important role of logic in the SW.

Logic-based languages express a domain specific ontology in terms of classes of objects as well as relevant relationships holding among such classes. These kinds of languages have formal well-defined semantics which are given via interpretations. An interpretation is a tuple (UI, *I), where UI is the domain of the interpretation and *I is a function that gives to all objects or combinations of objects and relationships, a meaning

in the domain UI. Given an ontology expressed in a logic-based language, a model is an interpretation that gives the ontology a meaning being consistent with respect to the domain UI. An ontology in such languages describes objects and relationships that have to exist and hold consistent meaning in all its possible models.

We will present languages based on first-order predicate logic. And then we discuss ontology languages based on Description Logics.

- **An ontology language based on first-order predicate logic**

For this kind of languages we consider the Knowledge Interchange Format (or briefly "KIF") language. An ontology in KIF is described in terms of constants, predicates, functions. KIF provides an important extension of first-order logic, by allowing the reification of formulas as terms used in other formulas. Therefore KIF allows meta-level statements. More in details, a knowledge base in KIF consists of a set of forms, where a form is either a sentence, or a rule, or a definition.

  - A sentence, which may be quantified universally or existentially, can be a logical constant, an equation, an inequality, a relation constant associated to an arbitrary number of arguments, or a combination of sentences by means of classic logical operators (negation, conjunction, disjunction, implication, and equivalence).
  - A definition can be viewed as shorthand for the sentences in the content of the definition, allowing to state sentences that are true by definition.
  - A rule, that is called forward or reverse depending on the particular operator involved, can be monotonic or non-monotonic.

- **An ontology language based on Description Logics**

Description Logics (or briefly "DLs") are a family of logic-based knowledge representation formalisms designed to represent and reason about knowledge of an application domain in a structured and well-understood way. The basic notions in DLs are concepts and roles, which denote sets of objects and binary relations, respectively. Complex concept expressions and role expressions are formed by starting from a set of atomic concepts and atomic roles. Thus, a specific DL language is mainly characterized

by constructors it provides to form such complex concepts and roles. DLs differ from their predecessors, such as semantic networks and frames, in that they are equipped with a formal, logic-based semantics. Such well-defined semantics and implemented powerful reasoning tools make DLs ideal candidates for ontology languages.

Some languages based on DLs are the language ALC, AL-Log, Carin, PSL, OWL, etc. More information about these languages can be found in the report by [Lenzerini et al., 2003].

2.1.6 Semantic Web and Agent Technology

Agent technology has been investigated for years. This technology has the potential to greatly impact our lives and work, and application areas where they can play a crucial role include e-commerce, grid computing, Semantic Web, etc.

From the user's perspective, a software agent is a program that is capable of assisting people and acting on their behalf. Typical properties associated with agents include autonomy, pro-activity, reactivity, and sociality. However depending on specific applications agents may come in different types and operate in various settings.

From a system perspective, an agent is a software object that,

- Is situated within an execution environment
- Possesses the following mandatory properties:
  - *Reactive*: senses changes in the environment and acts according to those changes
  - *Autonomous*: has control over its own actions
  - *Goal-driven*: is proactive
  - *Temporally continuous*: is continuously executing
- Probably possess any of the following orthogonal properties:
  - *Communicative*: able to communicate with other agents
  - *Mobile*: can travel from one host to another
  - *Learning*: adapts in accordance with previous experience
  - *Believable*: appears believable to the end user

By creating an environment like the SW where information is described explicitly in a machine-processable format, agent technology can contribute and promote their

potential ability. And the real power of the SW will be realized when ontology-annotated information is available and software agents are built to collect web contents from diverse sources, process information, and exchange results with each other. The SW, which integrates logical inference, knowledge representation, and intelligent software agent technologies, introduces the notion of SW agents. Such SW agents will receive some tasks from a user, seek information from web sources, communicate with other agents and systems, combine selected information and present answers to its user. SW agents may not replace totally the role of human user on the SW but they can perform complex tasks and assist to reduce human effort.

From our view, SW agents should make use of the following technologies.

- Metadata and annotations are used to describe information of web resources by using SW languages.
- Ontologies are used to assist in web searches, to interpret retrieved information, and to communicate with other agents.
- Logic is used for representing and processing retrieved information as well as for reasoning in order to draw conclusions.

2.1.7 Conclusion

We have given the reasons motivating the move from current Web to its next generation—the SW—and described the vision of the SW. Essential, important concepts and technologies such as ontology, logic, logic programming, and agent technology are also introduced. They basically motivate the research work in this thesis.

2.2 Previous Works and our Proposed Framework

2.2.1 Introduction

A wide range of SW languages like RDF, RDF Schema, DAML+OIL, and OWL have been proposed. In addition, a number of independent tools and APIs such as RDF parsers, OWL parsers, ontology editors, etc. that implement these technologies have been developed. These tools are used to help us build and maintain ontologies by using SW languages. In the SW applications often require multiple distributed ontologies for querying and reasoning with them. Many works with different approaches have been

proposed for processing information of SW ontologies. In the following sections we will consider various approaches related to object orientation, description logic, logic programming, and agent technology for handling ontologies. Shortcomings and gaps of these approaches will be discussed and basing on this analysis we then introduce our proposed framework in this thesis.

2.2.2 Various Approaches for Processing Semantic Web Information

Due to the fact that RDF Schema and OWL are used to define classes, instances, and their relationships, hence, they could be related to an object-oriented modeling concept. In this direction, several approaches tried to handle this semantics of the ontology languages by an object-oriented system. [Bassiliades et al., 2004] described DR-Device, a defeasible logic system for the SW, which was a deductive object-oriented knowledge base system which is implemented on top of the C Language Integrated Production System (or briefly "CLIPS") and on R-Device [Bassiliades & Vlahavas, 2004], a deductive rule system over RDF data. In another approach in this direction, [Koide & Kawamura, 2004] introduced an SW processor, called "SWCLOS", which was built upon the Common Lisp Object System (or briefly "CLOS"). According to this work, every resource in RDF and RDF Schema, e.g. `rdfs:Class`, `rdfs:Resource`, `rdf:Property`, resource instances, and properties are realized as CLOS objects with straightforward mapping RDF/RDF Schema classes to CLOS classes and RDF/RDF Schema instances to CLOS instances. Axioms and entailment rules in RDF/RDF Schema are embodied in the system. Apparently, both these approaches are heavily relied on traditional object-oriented concepts and were mainly concerned with representing and querying RDF data with an object-oriented system which is not a logical system.

There have been many research works paying attention to the problem concerned with multiple ontologies in the SW. A rather traditional approach proposed by [Fernandes et al., 2003] was used to develop an ontology-based knowledge system for creating and sharing user's personal objects, with support for querying semantic associations. A global ontology is available on a web server and provides global

semantic query capability. This ontology is used as an intermediate component for communication among users' local ontologies which are all translated to the global one.

There are several ways to represent SW information and each representation schema typically has associated software tools to manipulate information so long as it is represented properly. [Bowers & Delcambre, 2000] introduced their approach which was used to represent information for a wide variety of model-based applications in a uniform way by using RDF language, and to provide a mapping mechanism that was used to easily transform information from one representation to another.

In order to share information defined in ontologies in the SW, softwares that use different vocabularies must be able to translate data from one ontological framework to another. Ontology translation is required when translating datasets, generating ontology extensions, and querying through different ontologies. OntoMerge, an online system for ontology merging and automated reasoning proposed by [Dou et al., 2003], was implemented for ontology translation with inputs and outputs expressed in OWL or other web languages. In this approach, the merge of two related ontologies is obtained by taking the union of concepts and axioms defining them, and then adding bridging axioms that relate to their concepts. The resulting merged ontology then serves as an inferential medium within which the translation can occur. Their work was focused more on how to express semantic differences between two ontologies in a merged ontology, and how to implement ontology translation by inference. This approach can support for developing some semi-automatic tools for ontology merging but not fully automated.

Another approach related to ontology translation was presented by [Mota & Botelho, 2005]. They developed an ontology translation web service based on the O3F framework, an ontology representation framework [Mota et al., 2003], that translates ontologies between each other expressed in OWL. In their O3F framework, ontology translation was based on a mapping from basic concepts of the source ontology to basic or compound concepts of the target ontology. They also discussed in details the algorithms used by the ontology translation web service and this service's interface. An illustration of this approach was also presented in the work.

Paying an attention to the ways to represent the complex relationships between multiple ontologies, [Xu et al., 2004] analyzed and classified a variety of relationships

between multiple ontologies, then proposed a theory of bridge ontology. The bridge ontology in this approach was a special ontology that has four layer structures and is able to express twelve kinds of relations between multiple ontologies. A bridge relation was defined as an associated rule between different ontologies and can be applied for concepts or relations between these ontologies.

Concerning ontology merging, the method FCA-MERGE addressed by [Stumme & Maedche, 2001] used a bottom-up approach which offered a structural description of the merging process. This method was guided by application-specific instances of the given source ontologies that were to be merged. They also applied techniques from natural language processing and formal concept analysis for deriving a lattice of concepts as a structural result of the FCA-MERGE. The generated result was then explored and transformed into the merged ontology with human interaction.

The RDF language is considered to be an important relevant standard for data representation and exchange in the SW. Therefore, there are a number of languages for querying RDF documents having been proposed. Some of them relied on some traditional database query languages like RDQL [Seaborne, 2004], or on the Notation 3 syntax of RDF like N3 [Berners-Lee, 2000], [Roo, 2002], or on rule languages like Triple [Sintek & Decker, 2002] or on the graph model of RDF like RQL [Karvounarakis et al., 2002]. A key distinction of these approaches is the support for RDF Schema semantics. Languages like N3 and Triple [Miklos et al., 2003] do not make a strict distinction between queries and rules. Thus a logic program representing the desired semantics, in this case RDF Schema, can optionally supplement a query. RQL language supports for RDF Schema semantics internally while RDQL language ignores RDF Schema semantics. A more detail comparison of these query languages and others for RDF was presented in [Haase et al., 2004].

2.2.3 Logic Programming Approaches for Semantic Web

The Semantic Web is a promising application area of logic programming language like Prolog for its non-determinism and pattern-matching. [Wielemaker et al., 2003] described an infrastructure based on Prolog for handling large ontologies defined by using SW languages RDF and RDF Schema. The parser, storage, and basic query

interface for this RDF infrastructure were provided in terms of a library of Prolog predicates. This infrastructure can handle moderately large RDF triple sets with fast parsing, fast access but it depends on a Prolog interpreter to query information. Queries at RDF Schema were implemented by using trivial Prolog rules. The library supports almost querying rather than reasoning with ontologies, especially not used for OWL.

[Peer, 2002] introduced a logic programming approach for transforming RDF documents and RDF queries, in order to provide the amount of expressivity needed to build generic services for RDF data management. In this approach RDF statements are represented as logical expressions via ternary relations having the form of triple (Subject, Predicate, Object). An RDF document therefore can be represented as a set of ternary predicate triples. For queries, they are expressed as (Horn-) rules, which may be asserted to and executed by a logic programming based engine, e.g. a Prolog interpreter. Algorithms for transforming ontologies and queries were presented and a mapping mechanism for structural differences of mapped ontologies was also discussed in their work.

[Grosof & Horrocks, 2003] claimed that a mapping between ontology and rule languages was important for many aspects of the SW such as an SW architecture, querying ability, data integration, and SW service. Therefore a mapping of a Description Logic (DL) subset into logic programs (LP) which is suitable for evaluation with Prolog was introduced. This interaction of DL with LP called Description Logic Program completely covers RDF Schema and a fraction of OWL. They showed how (some of) statements of DL and DL based languages (such as DAML+OIL and OWL) as well as complex compound expressions were built up from atomic classes and properties using a variety of constructors corresponding to def-Horn statements (rules). They also discussed some kinds of typical inference in DL and LP, and how they can be represented in each other, i.e., in LP and DL respectively. As remarked by [Weithoener et al., 2003], logical database systems seem most suitable to combine LP with efficient and persistent data storage. However applying this approach for loading, storing, and evaluating ontologies in logical database systems showed some significant scalability deficits as well as representational drawbacks. Therefore, [Weithoener et al., 2003] developed a new mapping without above limitations which was called the "meta

mapping" approach. This approach is *meta* in the sense that it maps the LP subset of OWL into a higher representational level resulting in lower computational complexity and more representational flexibility. The basic idea of this approach is to convert OWL statements contained in an ontology into a set of facts reflecting the content of the ontology. A description and detail comparison between this approach and the approach in [Grosof & Horrocks, 2003] as well as some benchmarking results for both of them were depicted in their work.

2.2.4 Agent-based Approaches for Semantic Web

The integration of agent technology and ontologies as well as SW technologies can significantly affect the use of web services and the ability of extending programs to be able to perform efficiently complex tasks for users without or less human intervention [Hendler, 2001]. This section will briefly review various techniques, approaches and problems which combine SW and agent technologies.

- **Applying SW technologies for developing SW agents**

[Eberhart, 2002b] presented a detailed description of the OntoAgent framework which is used for the declarative specification of agents. This platform allows a software agent to be specified using standard markup languages such as RDF, RDF Schema, and RuleML [Boley et al., 2001]. The functions of agent's components as well as how these components can be represented by using SW markup languages were described in the work. They also introduced the communication subsystem of an agent which distinguishes between two basic types of messages: queries and information messages. Another work proposed by [Dietrich et al., 2004] also presented a general architecture rule based agents for the SW. An agent of this kind has its behaviors and/or its knowledge expressed by means of rules. Their agent architecture was based on knowledge and perceptions implemented by using SW languages. For example, RDF Schema was used for specifying the schema of an agent's mental state or its factual knowledge while the RuleML was used for specifying terminological and heuristic (intentional) knowledge of the agent.

We can realize that both these works were focused on applying SW technologies for representing and developing agent architecture rather than processing SW information by basing on agent technology.

- **Agent communication in Semantic Web**

In an open and heterogeneous environment like the SW, it is essential to enable agents communicate with one another to search for information and services. Many works have paid attention to problems of communication between agents in the SW. Researchers frequently make three assumptions to enable agent communication: sharing ontologies, using the same communication language, and pre-agreeing on a message format. However, in order to allow agents interoperate on the fly, [Al-Muhammed & Embley, 2004] presented an approach which enables communication among independent agents in the SW by assuming neither shared ontologies, nor a common language, and nor a shared message format. This approach was based on two key ideas. (1) *Independent global and local ontologies*. Rather than requiring agents to share ontologies, they provided an agent-independent, domain specific ontology, called the global ontology. When an agent wishes to communicate within a particular application domain, essential information from the agent's local ontology including class names, parameter names, variable names, and the types of these concepts will be mapped to the global ontology. (2) *Automatic message service mapping*. Rather than having agents deal directly with incoming messages, the system automatically maps an incoming message to an appropriate service. Therefore agents do not have to use the same communication language and pre-agree on a message format.

[Takeda et al., 1995] discussed how ontologies play roles in building a distributed and heterogeneous knowledge base system. The most important but difficult problem for agents sharing and reusing knowledge is how to use their different concept structures together. There are mainly two reasons for it. One is that it is difficult for each agent to know what concepts are used and to know which agent to talk to. The other is, even if they can find relationships among their concept structures, it is difficult for each agent interpreting concept structures of other agent. To solve these problems, they proposed that (1) providing ontologies which are containers to put concepts and their

relations used in each agent. Therefore each agent needs to declare its ontology explicitly. A mediation mechanism using ontology is used for suggesting possible agents to respond given undirected messages. (2) Allowing multiple aspects for concepts and providing relations among different aspects which are used to translate information from one aspect to the other. Although this approach has appropriate features for large-scale knowledge base systems, it is not sufficient to realize flexible communication among various heterogeneous agents.

[Baldoni et al., 2003b] presented various approaches for reasoning about conversations protocols within a framework of logic-based agent languages. They shown that a theory of communicative actions can be formulated in the Dy-Log logical framework, a high-level logic programming language for modeling and programming rational agents [Patti, 2002], [Baldoni et al., 2003a], so as to allow the modeling of software agents that can interact with one another by a speech act based communication mechanism. The framework allows an agent to reason about conversation protocols with other agents. They also presented an action theory based on the logic Dynamic Linear Time Temporal Logic [Henriksen & Thiagarajan, 1999] which provides a unified framework for specifying and verifying systems of communicating agents.

- Multi-agent systems in Semantic Web

Because of the promising vision of building multi-agent applications in the SW, there are also many works related to building multi-agent based applications using SW technologies. [Grimnes et al., 2003] developed a multi-agent application, called "GraniteNights", which allows a user to schedule an evening out. The application combines agent and SW technologies, being viewed as an agent-based SW service. There are various kinds of agents within this application infrastructure: information agents (wrapper for RDF resources), profile agent (manages user data, such as identification, password, preferences), constraint-solver agent (maps RDF data to finite domain constraints and produces valid instantiated schedules), and evening agent (receives user queries and invocation of other agents to generate a solution). This approach used SW languages like RDF as a content language for describing resources

of interest in an application domain and RDF Query by Example for expressing queries in RDF. Another scheduling application was developed by [Payne et al., 2002b]. The RETSINA Calendar Agent (Rcal) is a distributed meeting scheduling agent that performs scheduling with distributed ontologies about events, e.g. conferences, classes, which are published on the SW. In this application, translation services are used to transform concepts defined within previously unknown ontologies into known concepts, therefore allowing agents to understand the available information and achieve its goal.

Another work using SW technologies in multi-agent systems was presented by [Zou et al., 2003]. This approach used SW languages as content languages within the FIPA ACL messages; as the basic for agent knowledge bases via XSB-based reasoning tools; and to describe and reason about services. In another approach [Gandon et al., 2002] showed how some aspects of the underlying graph model of the SW framework could be exploited to handle allocation of annotations and distributed query solving in a multi-agent system. The approach concentrated on finding mechanisms to decide where to store newly submitted annotations and how to distribute a query in order not to miss answers when the needed information are split over several repositories.

2.2.5 Our Proposal: A Meta-logical Framework for Agent Communication of Semantic Web Information

SW is a promising area so there are a lot of works that have been done so far to motivate its vision. Although approaches for developing query languages for SW information have gained some success in querying RDF data but they almost do not pay much attention to the problem of reasoning or supporting for RDF schema and OWL languages. Therefore their abilities for exploiting SW ontology information are still limited.

For approaches related to multiple ontologies, they concentrated on some aspects concerning with translating information from one ontology to another or merging information from multiple ontologies. And again, these approaches do not mention much some problems of reasoning with multiple ontologies or distributed reasoning.

Agent-based approaches are used for either building domain specific applications like scheduling in order to exploit information resources described by SW languages or combining with SW technologies to propose SW agent architectures. In

our view, agent systems should have capability of reasoning with multiple distributed ontologies to form a distributed reasoning system for processing SW information.

Besides that logic programming approaches have focused on representing SW information as logical sentences and performing querying or simple reasoning with them by employing some available logic programming engines like Prolog interpreter.

Our proposal in this thesis is to fill these gaps of previous approaches. By utilizing meta-logic we have proposed a meta-logical representation for SW ontologies which enables our agent system reasoning with multiple distributed ontologies and exchange information with one another. The agents and their communication are also characterized in meta-logic. By exploiting benefits of agent technology and logic programming as well as their relations with SW technologies, our approach is to develop a meta-logical framework for agent communication of SW information. This framework will be described and depicted in further details by the following chapters.

2.2.6 Conclusion

We have introduced the SW and its related background. We also have given our survey of several works in the SW. The result of this study led us to our own direction of investigation of the SW as follows.

- A unified method for representing SW information in a logical system.
- A sufficient and efficient inference engine needs to be developed to reason with SW ontologies.
- Agent-based framework is a suitable approach for exchanging SW information in terms of explicit and implicit knowledge.
- It is essential to reason with multiple distributed ontologies.

Chapter 3

A Meta-logical Framework for Agent Communication of Semantic Web Information

Our meta-logical framework is briefly called "MAC-SWI Framework" and this name will be used henceforth in the thesis.

3.1 Overview of the MAC-SWI Framework

The MAC-SWI, depicted in Figure.3.1, is a meta-logical framework for agent communication of SW information, expressed in a meta-level logic programming.

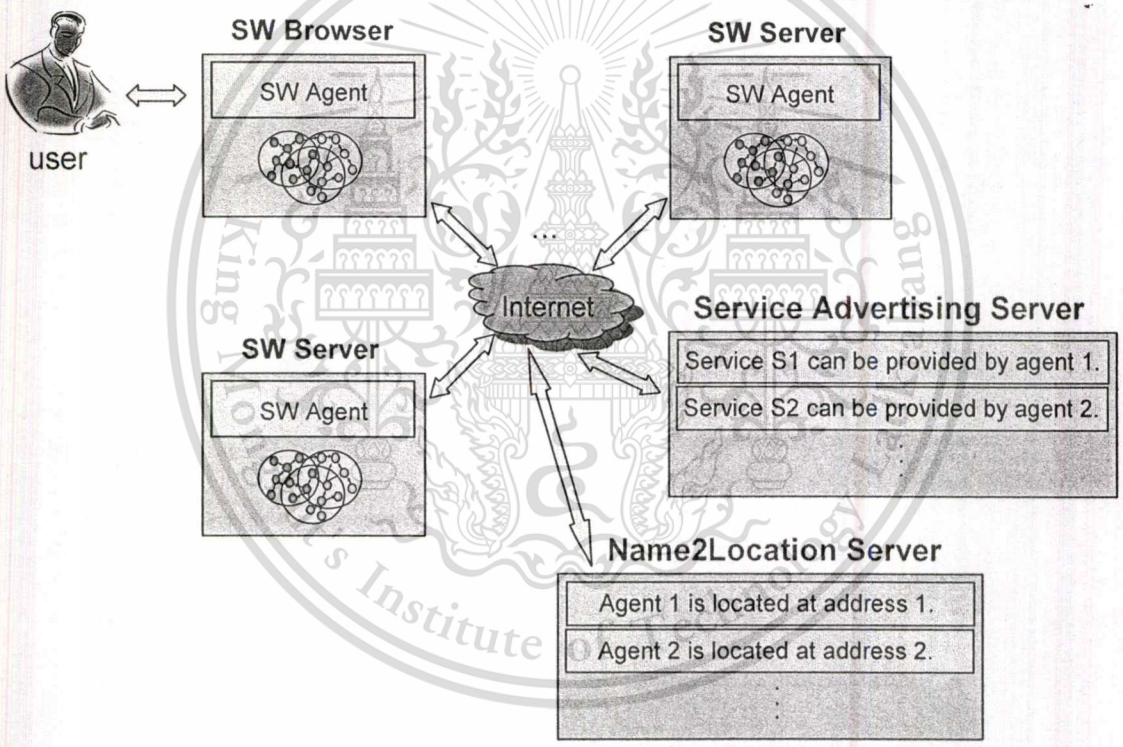


Figure.3.1 Multi-agent communication system for the SW

In order to concretely describe the framework, we focus on two main problems: SW information (ontologies) processing by one single agent and agent communication of SW information.

The meta-logical system for one single agent consists of three main parts: meta-programs of multiples ontologies, a meta-interpreter, and the communication facility.

Appendix II Configuration, Installation, and Deployment of the MAC-SWI Framework

II.1 Installation

- Install the SWI-Prolog 5.5.11 and make sure that the installation includes SGML/XML/HTML parser, RDF/SW Library.

  Website: *http://www.swi-prolog.org*

- Install Win-Prolog 4.320, including Intelligent Server.

  Website: *http://www.lpa.co.uk*

- Install Win-Prolog 4.040, including TCP/IP, HTTP, KQML, and Agent Libraries.

  Website: *http://www.lpa.co.uk*

- Install ProWeb server. After installing Win-Prolog 4.040, move the entire PROWEB directory into an Internet public directory that will be used to deploy servers.

  Website: *http://www.lpa.co.uk*

- Install Apache HTTP Web server 2.0

  Website: *http://www.apache.org*

- Install browsers like Internet Explorer 6.0, Firefox 1.5, Opera 6.0, etc.

  Websites: *http://www.microsoft.com*,
  http://www.mozilla.com,
  http://www.opera.com

- Testing each system to make sure that they work correctly. More information about installation process can be found in manuals which are supplied along with those software toolkits.

II.2 Configuration

In this section we will consider some instructions for configuring an Apache HTTP Web server, an important component for the whole system.

- Apache HTTP Web server 2.0

After installing the Apache server, we need configure some configuration directives that give the server necessary instructions. The configuration file *httpd* can be found in the Apache server directory *Apache Group/Apache 2/conf*. Some directives need to be configured with suitable values are domain name, server name, server admin, server port, server root, etc. The meaning and instructions for these directives can be referenced in the *httpd* file.

- *Alias*: Aliases are used to map URLs to file system locations. To add an alias of a server, add the following directives to the *httpd* file. According to specific requirements, we can include or add some options and instructions to these directives. In this example, we add an alias for the directory containing html files that can be accessed by clients.

```
Alias /html "C:/SemanticWeb/PROWEB/SWServer/HTML"
<Directory "C:/SemanticWeb/PROWEB/SWServer/HTML">
  Options FollowSymLinks
  MultiViews IncludesNoExec
  AddOutputFilter Includes html
  AllowOverride None
  Order allow, deny
  Allow from all
</Directory>
```

- *Script alias*: Script aliases are used to map URLs to file system locations and designate the targets as CGI scripts. To add a script alias, use the following sample.

```
ScriptAlias /ProWeb/ "C:/SemanticWeb/SWServer/"
<Directory "C:/SemanticWeb/SWServer/">
  AllowOverride None
  Options None
  Order allow,deny
  Allow from all
</Directory>
```

More information related to configuration of aliases and script aliases can be found at the address *http://httpd.apache.org/docs/2.0/mod/mod_alias.html*.

II.3 Deployment

- An SW browser can be deployed as a Win-Prolog application or as a plug-in of Firefox browsers.
- An SW server, a service advertising server, or a Name2Location server can run as a Win-Prolog application or can be deployed on Apache + ProWeb, or Intelligent server.
- A normal server is deployed on Apache server or Apache + ProWeb.

Each agent can be deployed on separate machines on the network or several agents can run on the same computer. Name2Location servers and service advertising servers are deployed first and they will continue waiting for requests of registration, advertisement, queries, etc. from SW servers, SW browsers. Then we can deploy SW servers as well as other normal servers. After that, the community of agents can work together. Some agents can leave and some can take part in the system during the time.

Appendix III Demonstrations for the MAC-SWI Framework

In this appendix, we will present three demonstrations to show the abilities of reasoning with multiple distributed ontologies of our agent framework. The first demonstration shows the powerful reasoning capability of our meta-interpreter with the family ontology. The second demonstration is to reason with multiple ontologies including two genealogy ontologies, the family ontology, and the cross ontology which is the linked ontology among previous ones. The third demonstration shows the full agent system working with multiple distributed ontologies in an e-commerce scenario.

III.1 Demonstration for Reasoning Capability: Family Ontology

We use the family ontology taken from the library of ontologies [Family_onto, 2003] for the demonstration. The ontology family is an OWL ontology describing the usual classes, e.g. `Person`, `Man`, `Woman`, `Child`, `Daughter`, `Parent`, `Father`, etc., and relationships, e.g., `hasParent`, `hasFather`, `hasChild`, etc. within a family. Each person may have some relationships to other persons, such as father, mother, brother, sister, uncle, aunt, etc. And there are many dependencies between those relationships. Information of a person can easily be represented in the ontology by OWL statements, e.g. `hasParent` relation is the inverse of `hasChild`, `Father` is a subclass of `Parent`. However we can use our SW meta-interpreter for exploiting many other relationships among people that are not defined explicitly in the ontology.

Figure.III.1 shows a part of the family ontology in its original OWL form which contains definitions of classes `Person`, `Man`, `Father`, `Mother`, `Son`, and relationships `hasChild`, `hasUncle`, `hasNephew`, `hasSibling`, `hasBrother`. Figure.III.2, Figure.III.3, and Figure.III.4 present graph models of property relationships as well as class hierarchy of classes and properties which are defined in the family ontology. Figure.III.5, Figure.III.6, and Figure.III.7 depict some parts of the MP meta-program, MMP meta-program transformed from the family ontology, and the query answering with our meta-interpreter, respectively. Each answer of the query answering is explained with essential clauses of the AMP meta-program (chapter 5), the meta-interpreter (chapter 6), and the family MP, MMP meta-programs from which the answer is derived.

```
<owl:Class rdf:about="#Person">
  <owl:equivalentClass>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:ID="Man"/>
        <owl:Class rdf:about="#Woman"/>
      </owl:unionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
<owl:Class rdf:about="#Man">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Restriction>
          <owl:onProperty>
            <owl:FunctionalProperty rdf:about="#hasSex"/>
          </owl:onProperty>
          <owl:hasValue rdf:resource="#Male"/>
        </owl:Restriction>
        <owl:Class rdf:about="#Person"/>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
<owl:Class rdf:ID="Father">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:ID="Parent"/>
        <owl:Class rdf:about="#Man"/>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
<owl:Class rdf:ID="Mother">
  <owl:equivalentClass>
```

```
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Parent"/>
        <owl:Class rdf:about="#Woman"/>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
<owl:Class rdf:ID="Son">
  <owl:disjointWith>
    <owl:Class rdf:ID="Daugther"/>
  </owl:disjointWith>
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Man"/>
        <owl:Class rdf:about="#Child"/>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>

<owl:ObjectProperty rdf:about="#hasChild">
  <rdfs:domain rdf:resource="#Person"/>
  <rdfs:range rdf:resource="#Person"/>
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#hasParent"/>
  </owl:inverseOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasUncle">
    <rdfs:domain rdf:resource="#Person"/>
    <rdfs:range rdf:resource="#Man"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasNephew">
  <rdfs:domain rdf:resource="#Person"/>
  <rdfs:range rdf:resource="#Man"/>
</owl:ObjectProperty>
```

```
<owl:ObjectProperty rdf:about="#hasSibling">
  <rdfs:domain rdf:resource="#Person"/>
  <rdf:type rdf:resource=
    "http://www.w3.org/2002/07/owl#SymmetricProperty"/>
  <rdfs:range rdf:resource="#Person"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasBrother">
  <rdfs:domain rdf:resource="#Person"/>
  <rdfs:subPropertyOf rdf:resource="#hasSibling"/>
  <rdfs:range rdf:resource="#Man"/>
</owl:ObjectProperty>
...
```

Figure.III.1 An OWL fragment of the family ontology

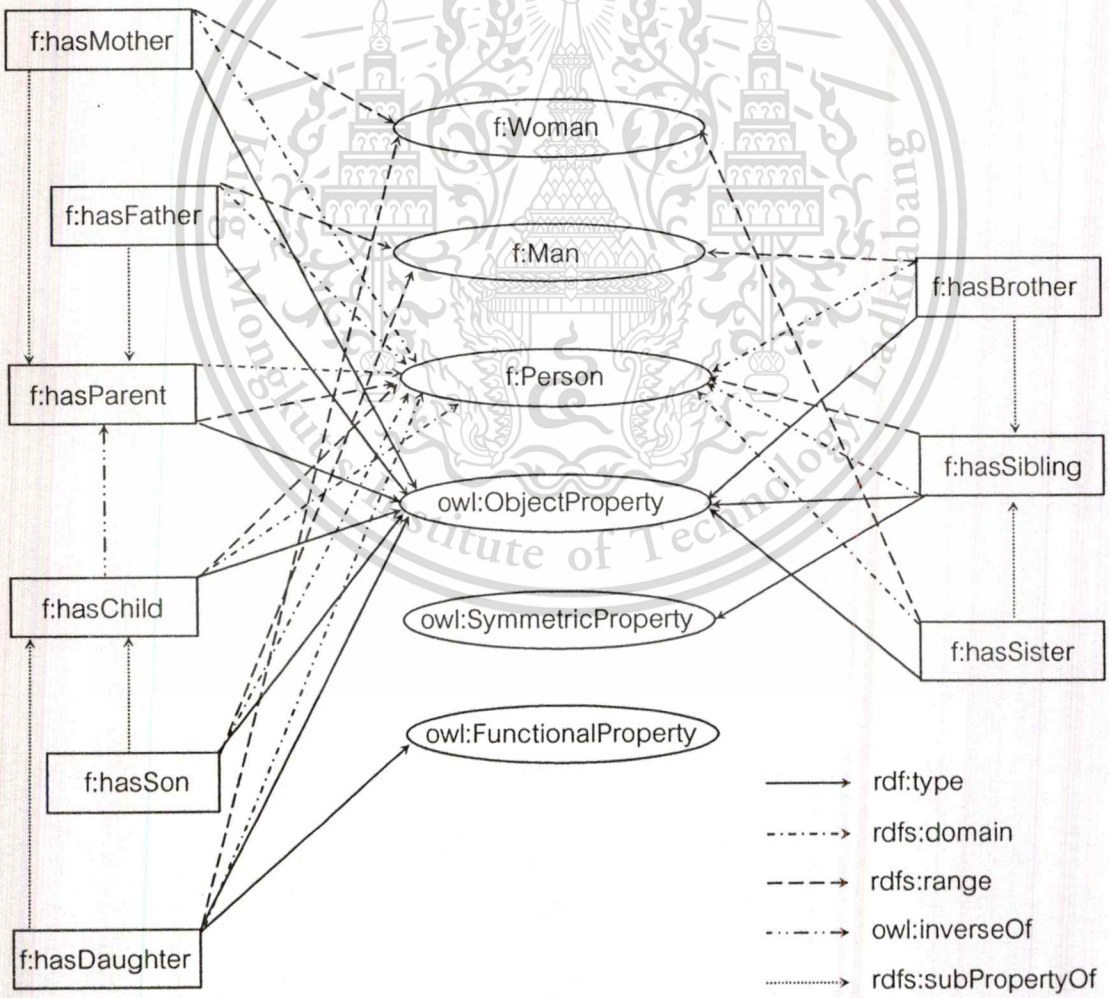


Figure.III.2 The property relationships of the family ontology (1)

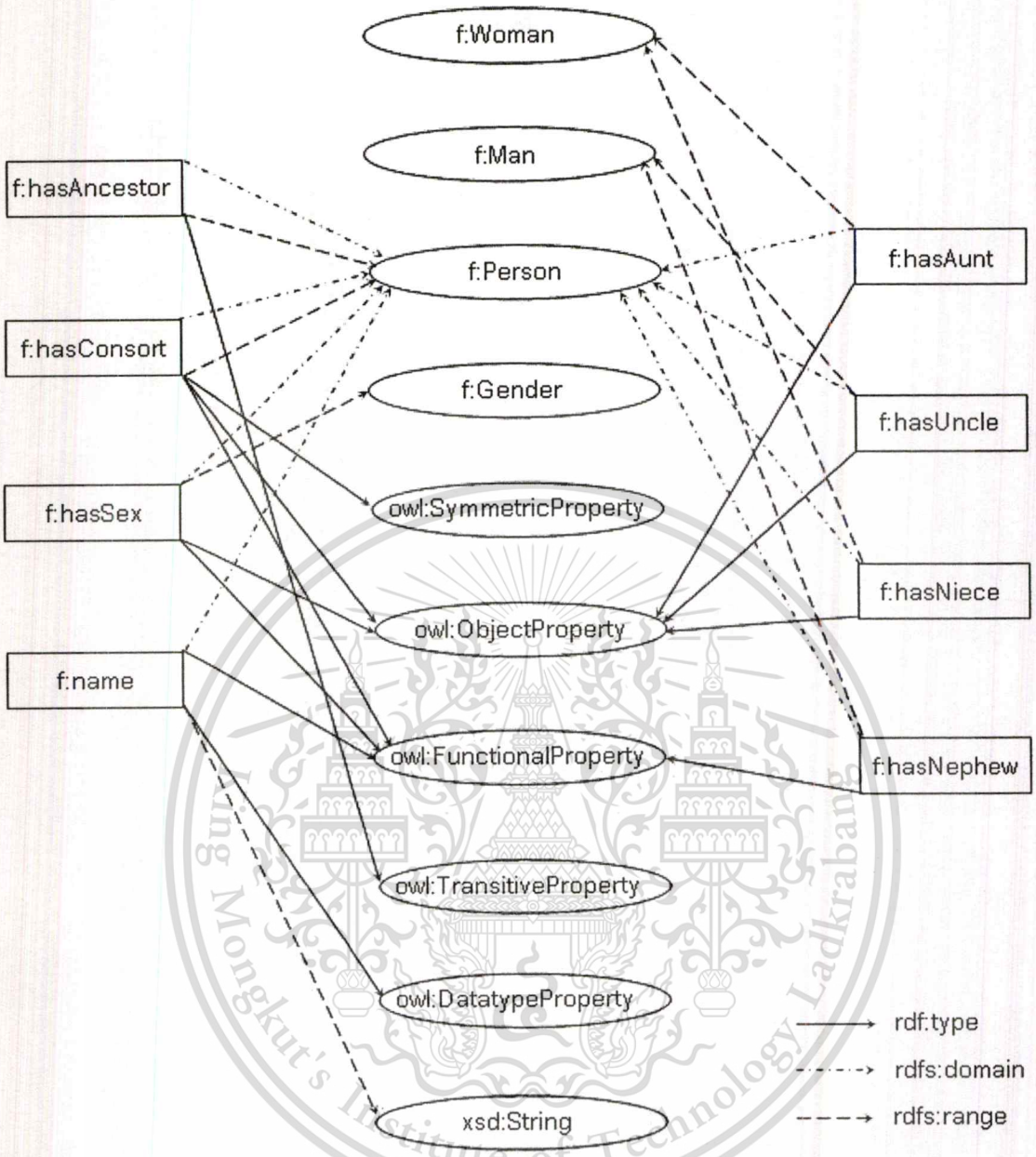


Figure.III.3 The property relationships of the family ontology (2)

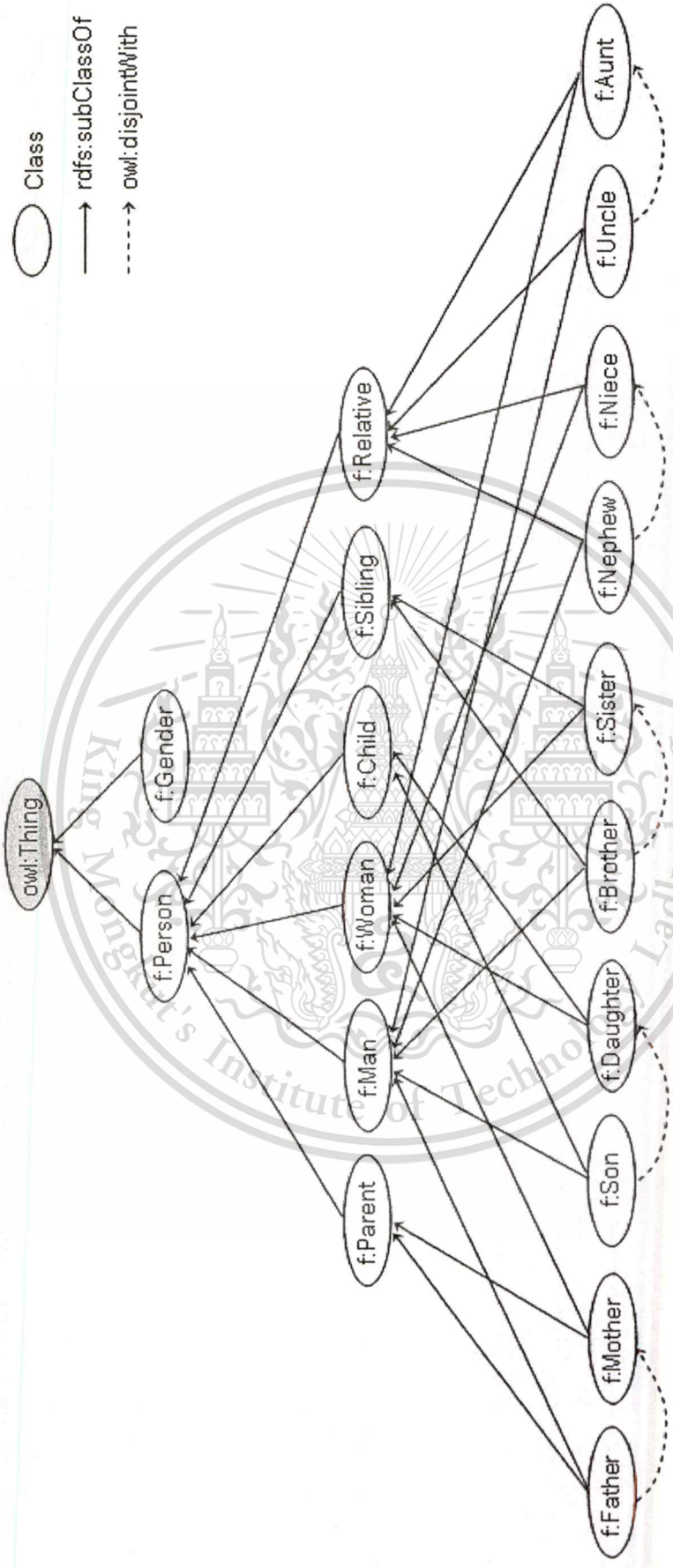


Figure.III.4 The class hierarchy of the family ontology

Due to the size of the ontology example, only some parts of the MP and MMP meta-programs transformed from the family ontology are shown below:

- The MP program:

```
statement(fo, 'fo'#'f':'hasParent'(                          (1')
   'fo'#'f':'M02', 'fo'#'f':'M01') ← true).
statement(fo, 'fo'#'f':'hasParent'(                          (2')
   'fo'#'f':'M03', 'fo'#'f':'M02') ← true).
statement(fo, 'fo'#'f':'hasBrother'(                         (3')
   'fo'#'f':'F02', 'fo'#'f':'M02') ← true).
...
```

Figure.III.5 MP program of the family ontology

- The MMP program:

```
meta_statement(fo, 'owl':'inverseOf'(                        (1)
   'fo'#'f':'hasParent', 'fo'#'f':'hasChild') ← true).
meta_statement(fo, 'rdfs':'subPropertyOf'(                   (2)
   'fo'#'f':'hasParent', 'fo'#'f':'hasAncestor') ← true).
meta_statement(fo, 'rdfs':'subPropertyOf'(                   (3)
   'fo'#'f':'hasBrother', 'fo'#'f':'hasSibling') ← true).
meta_statement(fo, 'owl':'symmetric'(                        (4)
   'fo'#'f':'hasSibling') ← true).
meta_statement(fo, 'owl':'transitive'(                       (5)
   'fo'#'f':'hasAncestor') ← true).
...
```

Figure.III.6 MMP program of the family ontology

Some query answering with the meta-interpreter:

```
?- demo(T, 'owl':'inverseOf'('fo'#'f':'hasParent', X)).
   T = fo, X = 'fo'#'f':'hasChild';
   // reasoning by clauses (mst), (true), (1)
?- demo(T, 'owl':'inverseOf'('fo'#'f':'hasChild', X)).
   T = fo, X = 'fo'#'f':'hasParent';
   // reasoning by clauses (asip), (ast), (mst), (true), (1)
```

```
?- demo(T, 'fo'#'f':'hasChild'('fo'#'f':'M01', X)).
   T = fo, X = 'fo'#'f':'M02';
   // reasoning by clauses (acip), (asip), (ast), (mst), (ost), (conj), (true), (1), (1')
?- demo(T, 'fo'#'f':'hasSibling'('fo'#'f':'F02', X)).
   T = fo, X = 'fo'#'f':'M02';
   // reasoning by clauses (acsp), (ast), (mst), (ost), (conj), (true), (3), (3')
?- demo(T, 'fo'#'f':'hasSibling'('fo'#'f':'M02', X)).
   T = fo, X = 'fo'#'f':'F02';
   // reasoning by  clauses (acsmp), (acsp), (ast), (mst), (ost), (conj), (true), (3), (4), (3')
?- demo(T, 'fo'#'f':'hasAncestor'('fo'#'f':'M02', X)).
   T = fo, X = 'fo'#'f':'M01';
   // reasoning by clauses (acsp), (ast), (mst), (ost), (conj), (true), (2), (1')
?- demo(T, 'fo'#'f':'hasAncestor'('fo'#'f':'M03', X)).
   T = fo, X = 'fo'#'f':'M02';
   // reasoning by clauses (acsp), (ast), (mst), (ost), (conj), (true), (2), (2')
   T = fo, X = 'fo'#'f':'M01';
   // reasoning by clauses  (actp), (acsp), (ast), (mst),(ost), (conj), (true), (2), (5), (1'), (2')
```

Figure.III.7 Query answering with the family ontology

III.2 Demonstration for Reasoning with Multiple Ontologies: Two Genealogy Ontologies, Family Ontology, and Cross Ontology

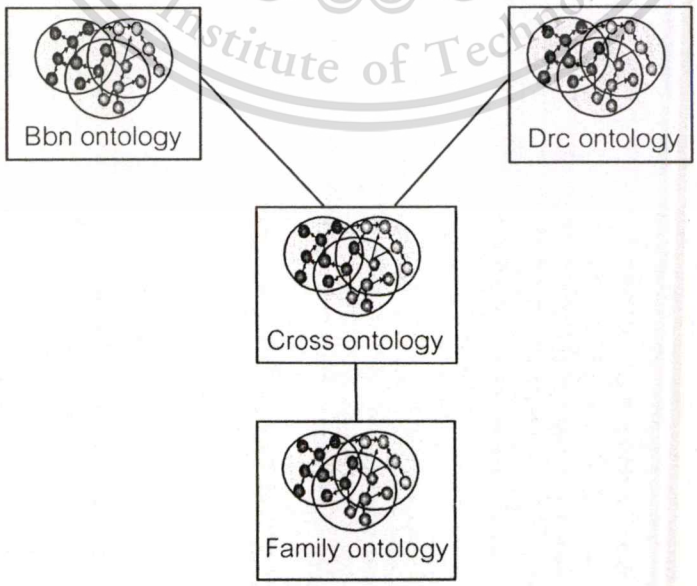


Figure.III.8 Multiple ontologies query answering demonstration

In this demonstration we use the two genealogy ontologies *bbn_ged* and *drc_ged* taken from [Bbn_ged, 2001], [Drc_ged, 2001], the family ontology *fo* [Family_onto, 2003], and a cross ontology *cross_onto* defined as a linked ontology of *bbn_ged*, *drc_ged*, and *fo* as an example of multiple ontologies for the demonstration. The overview of the demonstration is depicted in Figure.III.8. Some fragments of the *bbn_ged*, *drc_ged* ontologies are shown in Figure.III.9, Figure.III.10.

```
<b:Family rdf:about="#@F106@">
  <b:divorce rdf:resource="#event481"/>
  <b:marriage rdf:resource="#event482"/>
</b:Family>
<b:Family rdf:about="#@F1070@"/>
<b:Family rdf:about="#@F1074@">
  <b:divorce rdf:resource="#event2243"/>
</b:Family>
<b:Individual rdf:about="#@I1000@"
  b:name="Cicely" b:sex="F">
  <b:childIn rdf:resource="#@F373@"/>
  <b:spouseIn rdf:resource="#@F374@"/>
  <b:spouseIn rdf:resource="#@F375@"/>
  <b:birth rdf:resource="#event1280"/>
  <b:death rdf:resource="#event1281"/>
</b:Individual>
<b:Individual rdf:about="#@I1001@"
  b:name="Edward_V" b:sex="M">
  <b:childIn rdf:resource="#@F373@"/>
  <b:birth rdf:resource="#event1257"/>
  <b:death rdf:resource="#event1258"/>
</b:Individual>
<b:Marriage rdf:about="#event1025"
  b:date="15 DEC 1960" b:place="Brussels,Belgium"/>
<b:Birth rdf:about="#event1026"  b:date="1661"/>
<b:Death rdf:about="#event1027" b:date="1711"/>
<b:Divorce rdf:about="#event1041"/>
...
```

Figure.III.9 An OWL fragment of the Bbn ontology

```
<owl:ObjectProperty rdf:ID="childIn">
  <rdfs:domain rdf:resource="#Individual" />
  <rdfs:range rdf:resource="#Family" />
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="spouseIn">
  <rdfs:domain rdf:resource="#Individual" />
  <rdfs:range rdf:resource="#Family" />
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="birth">
  <rdfs:domain rdf:resource="#Individual" />
  <rdfs:range rdf:resource="#BirthEvent" />
</owl:ObjectProperty>
<owl:FunctionalProperty rdf:ID="location">
  <rdfs:domain rdf:resource="#Event" />
  <rdfs:range rdf:resource=
    "http://www.w3.org/2001/XMLSchema#string" />
</owl:FunctionalProperty>
<owl:Class rdf:ID="Family">
  <rdfs:label>Family</rdfs:label>
  <rdfs:subClassOf>
    <owl:Restriction owl:maxCardinality="1">
      <owl:onProperty rdf:resource="#husband" />
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction owl:maxCardinality="1">
      <owl:onProperty rdf:resource="#wife" />
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Event">
  <rdfs:label>Event</rdfs:label>
</owl:Class>
<owl:Class rdf:ID="FamilyEvent">
  <rdfs:subClassOf rdf:resource="#Event" />
  <owl:disjointWith rdf:resource="#IndividualEvent" />
</owl:Class>
```

```
<owl:Class rdf:ID="MarriageEvent">
  <rdfs:subClassOf rdf:resource="#FamilyEvent" />
</owl:Class>
<owl:Class rdf:ID="DivorceEvent">
  <rdfs:subClassOf rdf:resource="#FamilyEvent" />
</owl:Class>
...
```

Figure.III.10 An OWL fragment of the Drc ontology

In Figure.III.11, the **CMP** meta-program has meta-statements defining some rules for mapping the relationships among three ontologies, e.g., the first meta-statement `cf1` states that the property `'bbn'#'b':'place'` of the *bbn_ged* ontology and the property `'drc'#'d':'location'` of the *drc_ged* ontology are equivalent; the second meta-statement is a rule defining the relation `'fo'#'f':'hasFather'` (from the family ontology *fo*) based on the relations `'bbn'#'b':'sex'`, `'bbn'#'b':'spouseIn'`, and `'bbn'#'b':'childIn'` from the *bbn_ged* ontology. The **FMP** and **CMP** meta-programs are both required, as demonstrated in Figure.III.12, for supporting the meta-interpreter to perform reasoning in order to derive many implicit relations that are not defined explicitly in these ontologies.

The **FMP** meta-program contains meta-statements about the relationships (inversion, sub-property) and the characteristic (symmetric) of the properties `'fo'#'f':'hasParent'`, `'fo'#'f':'hasChild'`, `'fo'#'f':'hasBrother'`, and `'fo'#'f':'hasSibling'`.

BMP and **DMP** give information of some family relations using different terminology but these relations are somehow related to the family relations in **FMP**.

The demonstration of query answering is shown in Figure.III.12. According to the demonstration, the queries `q1` and `q2` are related to just the family ontology. That is, to answer `q1`, the interpreter can get the answer directly form **FMP**, but to answer `q2` the interpreter has to reason with the inverse characteristic of the `'fo'#'f':'hasParent'` property, the `'fo'#'f':'hasChild'` property, and some statements in **AMP**.

For queries `q3`, `q4`, and `q5`, some relationship definitions of the family ontology, i.e. the definitions of `'fo'#'f':'hasFather'` and `'fo'#'f':'hasSibling'`, are

used in the inferential process for answering the queries about the relationships of the individuals in the *bbn_ged* ontology; this is because the definitions of these relations and many others are not given in the *bbn_ged* ontology. To answer such queries, the interpreter has to perform reasoning with different ontologies. For the last two queries, q6 and q7, this demonstrates how to answer the queries about some relationships of individuals in the *bbn_ged* ontology by using some equivalent relations in the *drc_ged* ontology. The information telling what assumptions are used in the reasoning steps performed by the interpreter is also noted below each answer as seen in Figure.III.12.

CMP: Meta-program for the cross-referenced ontology

```
meta_statement(bbn u drc,                                    (cf1)
  'owl':'equivalentProperty'('bbn'#'b':'place',
  'drc'#'d':'location') ← true).
meta_statement(fo u T,
  'fo'#'f':'hasFather'(Child, Father) ←                      (cf2)
  demo(T, 'bbn'#'b':'sex'(Father, 'M')) ∧
  demo(T, 'bbn'#'b':'spouseIn'(Father, Family)) ∧
  demo(T, 'bbn'#'b':'childIn'(Child, Family))).
meta_statement(fo u T,
  'fo'#'f':'hasBrother'(Person, Brother) ←                   (cf3)
  demo(T, 'bbn'#'b':'sex'(Brother, 'M')) ∧
  demo(T, 'bbn'#'b':'childIn'(Brother, Family)) ∧
  demo(T, 'bbn'#'b':'childIn'(Person, Family))).
meta_statement(drc u T,
  'drc'#'d':'husband'(Husband, Family) ←                     (cf4)
  demo(T, 'bbn'#'b':'sex'(Husband, 'M')) ∧
  demo(T, 'bbn'#'b':'spouseIn'(Husband, Family))).
...
```

BMP: Meta-program for the Bbn_ged ontology

```
statement(bbn, 'bbn'#'b':'marriage'('bbn'#'b':'@F01@',
  'bbn'#'b':'event01') ← true).                              (b1)
statement(bbn, 'bbn'#'b':'place'('bbn'#'b':'event01',
  'Grafton Regis') ← true).                                  (b2)
statement(bbn, 'bbn'#'b':'sex'('bbn'#'b':'@I01@',
  'M') ← true).                                              (b3)
```

```
statement(bbn, 'bbn'#'b':'spouseIn'('bbn'#'b':'@I01@',
  'bbn'#'b':'@F01@') ← true).                                    (b4)
statement(bbn, 'bbn'#'b':'sex'('bbn'#'b':'@I02@',
  'M') ← true).                                                  (b5)
statement(bbn, 'bbn'#'b':'childIn'('bbn'#'b':'@I02@',
  'bbn'#'b':'@F01@') ← true).                                    (b6)
statement(bbn, 'bbn'#'b':'sex'('bbn'#'b':'@I03@',
  'F') ← true).                                                  (b7)
statement(bbn, 'bbn'#'b':'childIn'('bbn'#'b':'@I03@',
  'bbn'#'b':'@F01@') ← true).                                    (b8)
…
```

FMP: Meta-program for the family ontology

```
meta_statement(fo, 'owl':'inverseOf'('fo'#'f':'hasFather',
  'fo'#'f':'hasChild') ← true).                                  (f1)
meta_statement(fo,'rdfs':'subPropertyOf'(
  'fo'#'f':'hasBrother', 'fo'#'f':'hasSibling') ← true).(f2)
meta_statement(fo, 'owl':'symmetric'(
  'fo'#'f':'hasSibling') ← true).                                (f3)
…
```

DMP: Meta- program for the Drc_ged ontology

```
statement(drc, 'drc'#'d':'location'('drc'#'d':'event1138',
  'Dartford') ← true).                                           (d1)
…
```

Figure.III.11 MMP and MP programs of multiple ontologies demonstration

```
?- demo(T, 'owl':'inverseOf'('fo'#'f':'hasFather', X)).   q1)
   T = fo, X = 'fo'#'f':'hasChild';
   // reasoning by clauses (true), (mst), and (f1)
?- demo(T, 'owl':'inverseOf'('fo'#'f':'hasChild', X)).   (q2)
   T = fo, X = 'fo'#'f':'hasFather';
   // reasoning by  clauses (true), (mst), (ast), (asip), and (f1)
?- demo(T, 'fo'#'f':'hasFather'('bbn'#'b':'@I02@', X)). (q3)
   T = fo ∪ bbn, X = 'bbn'#'b':'@I01@';
   // reasoning by clauses (true), (ref), (ost), (mst), (ast), (conj), (cf2), (b3), (b4), and (b6)
```

```
?- demo(T, 'fo'#'f':'hasSibling'('bbn'#'b':'@I02@', X)).(q4)
   T = fo ∪ bbn, X = 'bbn'#'b':'@I03@';
   // reasoning by clauses (true), (ref), (ost), (mst), (ast), (conj), (acsp), (f2),(i3),(b5),(b6),(b8)
?- demo(T, 'fo'#'f':'hasSibling'('bbn'#'b':'@I03@', X)).(q5)
   T = fo ∪ bbn, X = 'bbn'#'b':'@I02@';
   // reasoning by clauses (true), (ref), (ost), (mst), (ast), (conj), (acsp), (acsmp), (f2), (f3),
   (cf3), (b5), (b6), and (b8)
?- demo(T, 'drc'#'d':'location'('bbn'#'b':'event01',X)).(q6)
   T = drc ∪ bbn, X = 'Grafton Regis';
   // reasoning by clauses (true), (ref), (ost), (mst), (ast), (conj), (asep), (acep), (cf1), and (b2)
?- demo(T, 'drc'#'d':'husband'('bbn'#'b':'@I01@', X)).   (q7)
   T = drc ∪ bbn, X = 'bbn'#'b':'@F01@';
   // reasoning by clauses (true), (ref), (ost), (mst), (ast), (conj), (cf4), (b3), and (b4)
```

Figure.III.12 Query answering with multiple ontologies demonstration

III.3 Demonstration for an E-commerce Scenario: The Bookshop Purchase

In this demonstration, we use a book purchase scenario of multi-agent communication that works with multiple distributed ontologies and distributed reasoning. Suppose we have an online bookshop selling books supplied by some publishers. The bookshop and the publishers have their own SW servers which provide information about the books able to be supplied by them. This book information is described by some SW ontologies and there are differences between the ontologies in the SW servers of different book shops and different publishers. The overview of the bookshop purchase demonstration is shown in Figure.III.13.

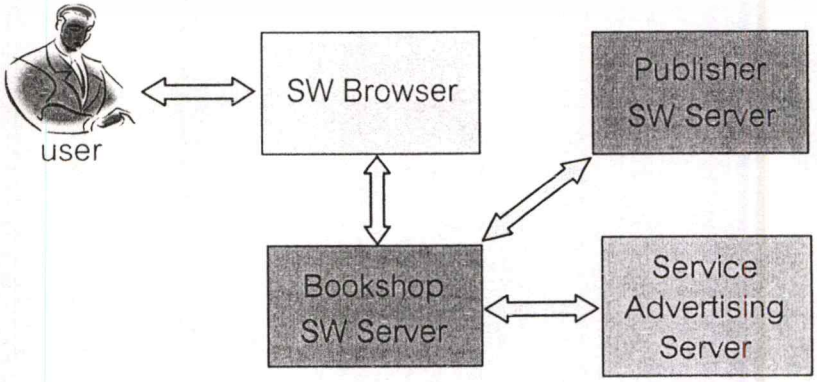


Figure.III.13 Bookshop purchase demonstration

An online book purchase begins with firstly a customer wants to buy some books from the bookshop. He then uses an SW browser to get some book information—i.e. title, short description about the book—(expressed in some ontologies) from the bookshop SW server. This information helps him decide which titles to buy. Sometimes, he may want to get more information of the interested titles, such as their publishers, book cover types (e.g. paperback, hardcover), and prices before placing an online order with the bookshop server. Suppose this information is not stored in the bookshop server, but the server can request it from some (probably unknown) publisher servers. In Figure.III.14, we present some parts of the meta-programs, MP and MMP, possessed by a publisher server, the bookshop server, and service advertising information in a service advertising server, respectively.

Publish SW Server

PMP: Meta-program for the publication ontology

```
statement(pmp, 'pmp'#'p':'bPrice'(
  'pmp'#'p':'0262631857', '$40') ← true).
statement(pmp, 'pmp'#'p':'bCover'(
  'pmp'#'p':'0262631857', 'hard') ← true).
statement(pmp, 'pmp'#'p':'bPrice'(
  'pmp'#'p':'0262635828', '$27') ← true).
statement(pmp, 'pmp'#'p':'bCover'(
  'pmp'#'p':'0262635828', 'paper') ← true).
```

Bookshop SW Server

BMP: Meta-program for the book ontology

```
meta_statement(bmp, 'rdf':'type'('Genetic Algorithm',
  'bmp'#'b':'GeneticProgramming') ← true).
statement(dmp u T, 'dmp'#'d':'bookInfo'(Title,Cover,Price) ←
  demo(bookShopAgent, T,
    'dmp'#'d':'bookInfo'(Title, Cover, Price))).
```

DMP: Meta-program for the documentation ontology

```
statement(dmp, 'dmp'#'d':'bTitle'('pmp'#'p':'0262631857',
  'Genetic Algorithm') ← true).
statement(dmp, 'dmp'#'d':'bTitle'('pmp'#'p':'0262635828',
```

```
  'Genetic Algorithm') ← true).
statement(dmp u pmp,
  'dmp'#'d':'bookInfo'(Title, Cover, Price) ←
    'dmp'#'d':'bTitle'(ISBN, Title) ∧
    demo(_, pmp, 'pmp'#'p':'bCover'(ISBN, Cover)) ∧
    demo(publisherAgent, pmp,
      'pmp'#'p':'bPrice'(ISBN, Price))).
```

Service Advertising Server

```
agentCapability(
  publisherAgent, channel('www.BookPublisher.com', 2000),
  service(pmp, 'pmp'#'p':'bCover'(ISBN, Cover))).
```

Figure.III.14 MMP and MP programs of the Bookshop purchase demonstration

```
?- demo(browser, _, 'rdf':'type'(X,
     'bmp'#'b':'GeneticProgramming')).
   X = 'Genetic Algorithm'
?- demo(browser, _, 'dmp'#'d':'bookInfo'(
     'Genetic Algorithm', Cover, Price)).
   Cover = 'hard', Price = '$40';
   Cover = 'paper', Price = '$27'
```

Figure.III.15 Query answering with the Bookshop purchase demonstration

A demonstration of the query answering of the SW browser is shown in Figure.III.15. To answer the first query, the SW browser reasons with its ontologies obtained from the bookshop server. However, for the second, the browser adopts BMP's the second statement, and this requires it to pass this query to the bookshop server to answer. The bookshop server uses DMP's third statement to infer the ISBN from the title; and it then queries an unknown publisher and the publisher *publisherAgent* for the cover type and price respectively. According to this example, since the 'Genetic Algorithm' book has two editions, i.e. hardcover and paperback, the user may ask to get more answers after the first one is derived and the browser then performs backtracking to get further answers with the similar reasoning steps of the previous one.

Appendix IV Author Biography

1	Personal Information						
1.1	Family Name	TRAN XUAN					
	Given Name(s)	VUONG					
1.2	Nationality	Vietnamese					
1.3	Date of Birth	Date	25	Month	Jan	Year	1980
1.4	Gender	Male	√	Female			

2	Academic Qualifications (I)				
2.1	Bachelors' Degree	Bachelor of Computer Engineering			
2.2	Program/Faculty	Faculty of Information Technology			
2.3	University	Ho Chi Minh City University of Technology, Ho Chi Minh City, Vietnam			
2.4	Year of Start	Month	Sep	Year	1998
2.5	Year of Completion	Month	Jan	Year	2003
2.6	Grade Point Average	3.35			(4-scale grading system)

3	Academic Qualifications (II)				
3.1	Master's Degree	Master of Computer Engineering			
3.2	Program/Faculty	Department of Computer Engineering, Faculty of Engineering			
3.3	University	King Mongkut's Institute of Technology Ladkrabang, Bangkok, Thailand			
3.4	Year of Start	Month	Jun	Year	2004
3.5	Year of Completion	Month	Mar	Year	2006
3.6	Grade Point Average	3.93			(4-scale grading system)

4	Employment Record

- Worked as an internship at the end of my bachelor study for four months in winter 2002 and as a software engineer in ELCA company for 14 months (February 2003 – April 2004). The general director was Mr. Jean-Paul Tschumi (Nationality: Swiss).
- Worked in some projects such as LEAF Framework (based on J2EE), Secutix (printing tickets at home), content management, electronic commerce.
- Participated in training newcomers for some projects.

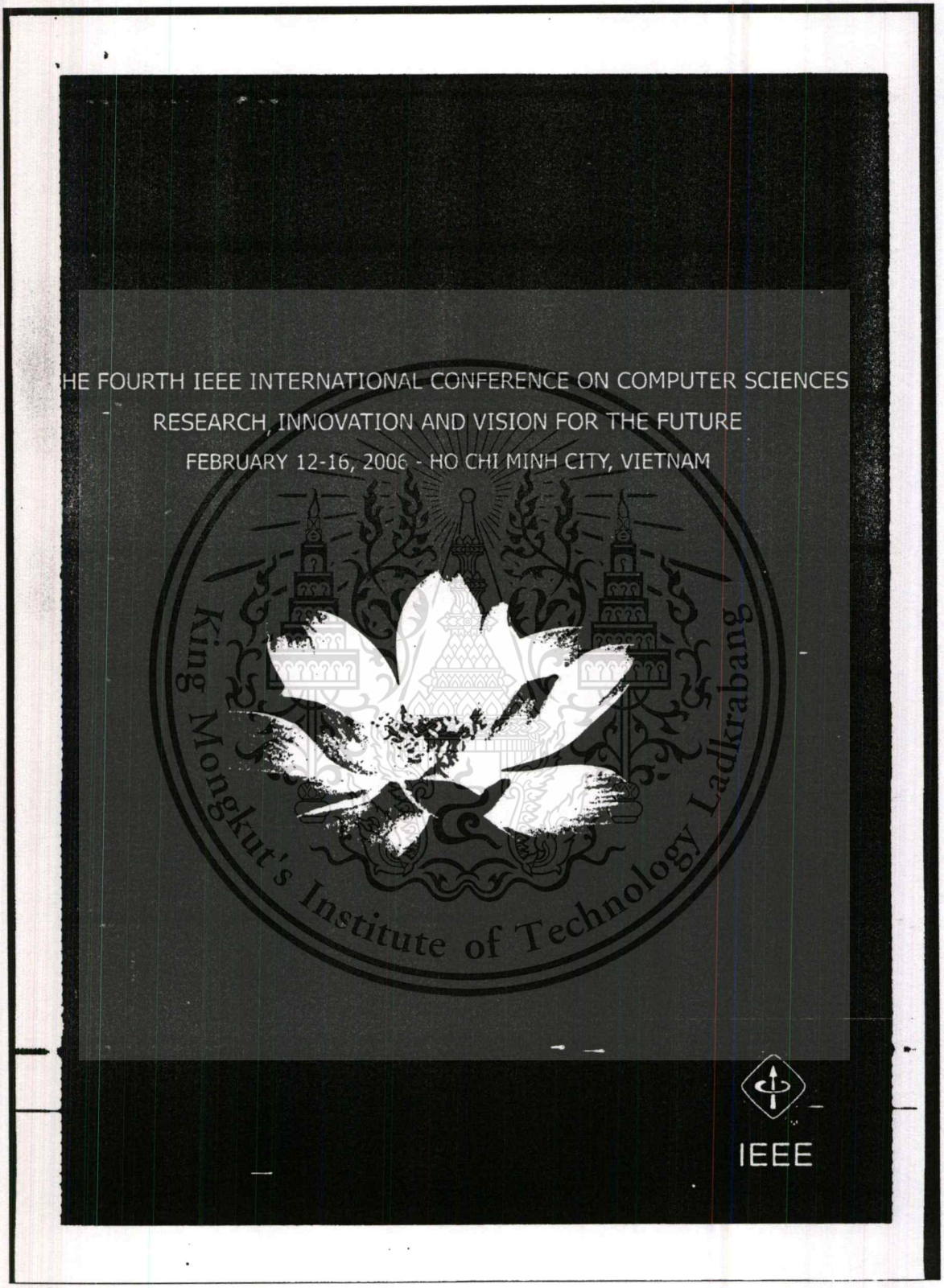
5	Publications

1. A Meta-logical Approach for Reasoning with Semantic Web Ontologies. In *proc. of the 4th IEEE International Conference on Computer Sciences: Research, Innovation & Vision for the Future*, February 12-16, 2006, HCM City University of Technology, HCM City, Viet Nam, pp. 228-235 (2006).

2. Meta-reasoning with Multiple Distributed Ontologies on the Semantic Web. In *proc. of the 8th International Conference on Intelligent Technologies*, December 14-16, 2005, Assumption University, Phuket, Thailand, pp. 301-309 (2005).

3. A Meta-logical Approach for Multi-agent Communication of the Semantic Web Information. In *proc. of the 16th International Conference on Application of Declarative Programming and Knowledge Management*, October 22-24, 2005, Waseda University, Fukuoka, Japan, pp. 7-16 (2005).

4. Semantic Web Agent Communication Capable of Reasoning with Ontology and Agent Locations. In *proc. of the 8th International Conference on Cybernetics, Informatics, and Systemics*, December 16-18, 2005 Krakow, Poland, pp. 98-104 (2005).



IEEE International Conference on Computer Sciences - RIVF'06
February 12-16, 2006, Ho Chi Minh City, Vietnam

A Meta-logical Approach for Reasoning with Semantic Web Ontologies

Visit Hirankitti and Vuong Tran Xuan

Abstract—**The design of Semantic Web markup languages, such as RDF, RDFS, and OWL, has been influenced by many different concepts, such as semantic structures and frame representations, object-oriented approaches, Description Logic, and first-order logic. Currently a Semantic Web ontology represented in these different languages is processed and manipulated differently by various approaches; however, in this paper we propose a single framework using meta-logic which can process the ontology in a uniform manner. Meta-logic allows us to reason with statements about description of classes, properties, instances, and their relationships expressed in RDF, RDFS, and OWL. This is accomplished by: initially an ontology, represented in combined RDF, RDFS, and OWL languages, is transformed to meta-logical statements representing definitions of classes and properties, relationships between classes and properties, and relationships between instances, etc.; later these statements are reasoned by a meta-interpreter which provides a query answering mechanism to infer meta and context information from the Semantic Web ontology.**

Index Terms—**Semantic Web, Ontologies, Meta-logic, Logic Programming**

I. Introduction

Ontology has been an essential part of the Semantic Web (or briefly "SW") as it describes the domain of discourse. Ontology is used to represent substantial human knowledge on the web. SW resources are described by ontology-based metadata formulated in several XML-based markup languages. For instance, RDF provides a basic data modeling in the form of subject-predicate-object triples; RDFS, OWL, and DAML+OIL [1] are used to define classes, properties, and their relationships. The development of these languages has been influenced by different concepts such as semantic structures and frame representations, object-oriented approaches, Description Logic, and first-order logic. A graph representation of RDF can be understood as a semantic net [2] or a conceptual graph [3]. An object-oriented approach also has an influence on the design of OIL [4], a part of DAML+OIL language from which OWL was derived. In addition, both DAML+OIL and OWL are developed based on Description Logic, a subset of first-order logic. Especially OWL DL, a sub-language of OWL, is designed to support Description Logic.

Due to the fact that several different concepts had influences on the design of these languages, this lead to the development of several different approaches for processing and manipulating SW ontologies. In this paper we shall consider some of those approaches that are based on logic programming and object-orientation before we characterize our meta-logical approach.

Some approaches based on logic programming transformed an SW ontology into a logic program in order to process and query with it [5]. In [6] an SW ontology represented in RDF, RDFS, and OWL languages was translated to predicates having the triple form and some rules were defined to transform RDF documents and RDF queries between each other. However, there was not any logical framework be proposed in both works.

Due to the fact that RDFS and OWL are used to define classes, instances, and their relationships, hence, they could be related to an object-oriented modeling concept. In this direction, several approaches try to handle this semantics of the ontology languages by an object-oriented system. For example, DR-Device [7] was a deductive object-oriented knowledge-based system which imported RDF data into the CLIPS production rule system as objects and used a deductive rule language for querying and reasoning about them. Another similar approach, an SW processor called "SWCLOS" [8], was built upon the Common Lisp Object System (CLOS). According to this approach, every resource in RDF and RDFS was mapped to CLOS objects. Apparently, both approaches relied heavily on traditional object-oriented concepts and were mainly concerned with representing and querying RDF data with an object-oriented system which is not a logical system. It seems that either direction still suffers from its own limitation. A logical approach basically does not support the concept of objects and classes, whilst an object-oriented approach does not support logical reasoning. To overcome such limitations while still adopting a logical approach, therefore, we shall propose a unified approach for manipulating SW information based on meta-logic.

The rest of this paper is organized as follows. Section II establishes our position, that is, to adopt meta-logic to reason with SW ontologies. Section III presents our meta-logical framework and section IV describes ontology meta-programs

This research work was supported by the Japan International Cooperation Agency (JICA) under the ASEAN University Network / Southeast Asia Engineering Education Development Network (or AUN/SEED-Net) Project.

Visit Hirankitti is with the Department of Computer Engineering, Faculty of Engineering, King Mongkut's Institute of Technology Ladkrabang, Ladkrabang Dist., Bangkok 10520, Thailand (corresponding author to provide phone: +66-2-739-2400; fax: +66-2-739-2404; e-mail: visit@ce.kmitl.ac.th).

Vuong Tran Xuan is with the Department of Computer Engineering, Faculty of Engineering, King Mongkut's Institute of Technology Ladkrabang, Ladkrabang Dist., Bangkok 10520, Thailand (corresponding author to provide phone: +66-2-739-2400; fax: +66-2-739-2404; e-mail: txvuong@yahoo.com).

IEEE International Conference on Computer Sciences - RIVF'06
February 12-16, 2006, Ho Chi Minh City, Vietnam

transformed from SW ontologies. Section V describes an SW meta-interpreter and Section VI shows how to employ it to query and reason with an SW ontology. Section VII covers some implementation issues. Section VIII compares our work with some other related works. Finally, section IX concludes this paper.

II. The Semantic Web with Logic and Objects

In this section, we explain the reason supporting our proposal where we adopt meta-logic to develop a unified approach to handle both logic and object concepts in SW ontologies.

A. The Semantic Web and Logic Programming

SW languages and logic programming languages are similar in that they are both declarative. In logic programming, a domain of discourse is described in term of objects and their relationships together with rules for deduction. In SW, just a part of information of resources is explicitly described. Given some ontologies and rules, implicit knowledge can be derived by using an inference engine. Hence, a logical system should be used for this purpose. Furthermore, ontology languages like DAML+OIL and OWL are heavily influenced by Description Logic which is a subset of first-order logic. To exploit this feature of SW languages a mapping—with some restrictions—from Description Logic to a Horn-clause program was proposed in [9].

To adopt logic programming for the SW, [5, 6] first transformed information of an SW ontology to a logic program and then defined rules to manipulate and reason about it. Alternatively, with logic programming we can use meta-logic to build a meta-interpreter to analyze, manipulate, and reason with the logic program transformed from the ontology. This ability is essential for the SW, since with the meta-logic we can develop a browser-like agent to reason about the SW ontologies and such an agent can communicate the ontologies among other agents. Even more importantly, the agents can reason about beliefs and truths based on those ontologies.

B. The Semantic Web and Object-oriented Modeling

Object-oriented concepts have been only partially adopted in ontology languages, like RDF, RDFS, and OWL. A class is used to define abstraction of related resources. However, class properties are not inherited automatically to its objects, unlike a traditional object-oriented approach where a class also defines properties (attributes) and methods and these will be inherited by its descendants and its objects.

In our framework, relations among objects are treated as logical statements and they are not encapsulated inside the objects. Our framework adopts partially an object-oriented concept only to manage ontology meta-information, like class hierarchy, property hierarchy, instance-class relations, etc.

C. A Meta-logical Framework for the Semantic Web

One way to reason with SW ontologies logically as well as with their classes, instances, and properties is to use meta-logic. An agent framework based on meta-logic can be developed in order to communicate, manipulate, and reason with SW information. Our research is moving towards this direction.

III. Our Meta-logical Framework

Our framework forms a logical system consisting of a meta-program and an inference engine. The former is in the form of logical sentences representing a meta-level description of an SW ontology. That is the ontology expressed in its naive form, e.g. in RDF, RDFS, and OWL languages, is transformed into a meta-logical representation. The latter is a meta-interpreter, in the form of a demo (meta-)program, which is used to infer explicit and implicit information, or in other words draw conclusions, from the former. The meta-interpreter could be extended to communicate to the Internet to obtain SW ontologies, communicate with the user to get SW information, draw inference consequences for the user, and traverse a link to an SW or web resource like a web browser.

Our meta-logical system for the SW can be simply depicted in the following diagram. Firstly, an SW ontology is retrieved from the Internet and is then transformed into a meta-program. The user can query the meta-interpreter which in turn derives answers from the transformed meta-program.

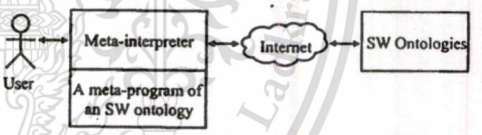


Fig. 1. Our meta-logical system for SW

To explain our framework, in the following we first develop two meta-languages of an SW ontology by mapping the language elements of RDF, RDFS, and OWL into meta-language terms, and then give an example of a meta-program that transformed from a family ontology [10]. Finally, we give details of our meta-interpreter together with its demonstration on query answering.

A. Language Elements of an SW Ontology

The language elements of an SW ontology are classes, properties, instances, and relationships between/among them described in the object level and the meta-level as illustrated in Fig. 2.

IEEE International Conference on Computer Sciences - RIVF'06
February 12-16, 2006, Ho Chi Minh City, Vietnam

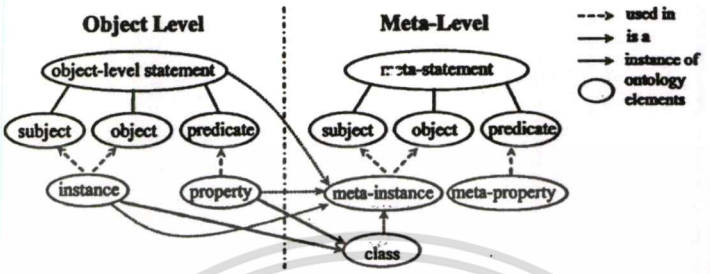


Fig. 2. The elements of an SW ontology at the object level and the meta-level

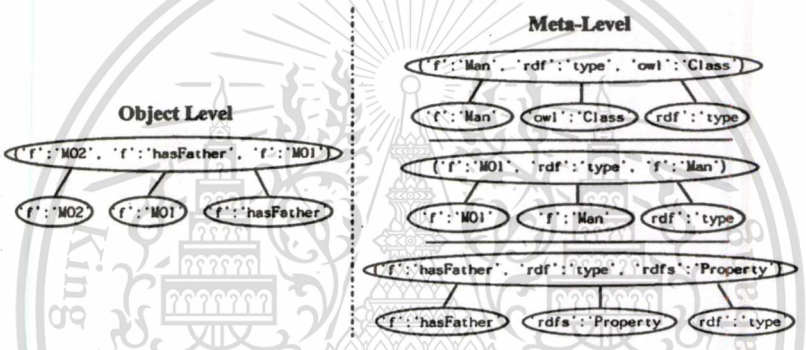


Fig. 3. Example of some elements of an SW ontology at the object level and the meta-level

At the object level, an instance can be an individual or a literal of a domain, e.g. `'john'`, and a property is a relationship between individuals or an individual's attribute, e.g. `'hasFather'`, `'name'`. At the meta-level, a meta-instance can be an individual, a property, a class, and an object-level statement. A meta-property is a property to describe a meta-instance's attribute or a relationship between and among meta-instances, e.g. `'subClassOf'`, `'equivalentProperty'`, etc.

To ease the understanding, some parts in the family ontology [10] are taken here to illustrate of the language mapping and the mapping is depicted in Fig. 3. In this example, the ontology contains a statement of the object-level and three statements of the meta-level. The first statement says that a man `'f':'MO2'` has the father `'f':'MO1'`. The second statement declares that a class `'f':'Man'` is an OWL class. The third statement declares `'f':'MO1'` as an instance of the class `'f':'Man'`. Similarly, the fourth statement declares that the property `'f':'hasFather'` is an instance of the class `'rdfs':'Property'`.

Notice that according to the SW convention, to make a name appearing in an ontology unique, we qualify it with a namespace like `<namespace>:<name>`, for example, `'f':'MO1'`, `'f':'hasFather'`, `'rdfs':'Property'`, `'owl':'Class'`, etc. Henceforth this qualified name will be used throughout.

Next we define meta-logical terms to express these elements of the ontology languages at the object level and the meta-level.

B. *Meta-languages of Semantic Web Ontologies*

In our meta-logical framework we distinguish between the object language and the meta-language of SW ontologies. The object language describes information of instances and their relationships in the real world. The meta-language describes the syntactic element of the object language. For example, in the real world John is the father of Tom, but at the meta-language "John" and "Tom" are two names, which contain four and three letters, respectively, representing nouns in the sentence.

Since both object-level and meta-level statements are to be reasoned (and probably manipulated) by a meta-interpreter, only their *syntactic forms* are processed by it. Thus, a meta-representation of the two kinds of statements is required by the meta-interpreter in order to reason with them. This is the reason why both object-level and meta-level statements here have to be expressed in yet another meta-language.

An SW ontology contains information of both object level

IEEE International Conference on Computer Sciences - RIVF'06
February 12-16, 2006, Ho Chi Minh City, Vietnam

and meta-level and this information is expressed by our meta-languages. More precisely, we classify the meta-language elements into 2 groups: one discusses about instances and their relationships, we call it "meta-language for the object level (ML)", and the other, called "meta-language for the meta-level (MML)", discusses classes, instances, properties, and their relationships.

- **A meta-language for the object level (ML)**

Instances and their relationships at the object level are specified in an ontology and this information is expressed at the meta-level by: meta-constants, meta-variables, meta-function symbols, meta-terms, and meta-statements as follows.

Meta-constant specifies mainly a name of an instance and a literal, e.g. `'john'`, and also a namespace. The latter is a meta-constant of MML. This means that ML and MML are not entirely separated.

Meta-variable stands for a different meta-constant at a different time, e.g. `Person`.

Meta-function symbol stands for a name of a relation between instances, or a name of an instance's property—i.e. an object-level predicate name, such as `'fatherOf'`, `'name'`— and also stands for other meta-level function symbol, e.g. `'←'`, `'∧'`, `':'`.

Meta-term is either a meta-constant or a meta-variable or a meta-function symbol applied to a tuple of meta-terms, e.g. `'f':'john'`.

To express an object-level predicate, it has the form: `P(S, O)` where `P` is an object-level qualified predicate name, `S` and `O` are meta-constants or meta-variables or meta-terms, e.g.

```
'f':'fatherOf'('f':'M2', 'f':'M1').
```

The meta-term expressing an *object-level sentence* is a term `P(S, O)` or a logical-connective function symbol applied to the tuple of these terms. We presume all meta-variables appearing in the object-level sentence are universally quantified. One form of the sentence is a Horn-clause, e.g.

```
'f':'fatherOf'(F, Ch) '←'
  'p':'parentOf'(F, Ch) '∧' 'm':'male'(F).
```

Meta-statement for the object level reflects an object-level sentence to its existence at the meta-level. It has the form: `statement(object-level-sentence)`, e.g.

```
statement('f':'fatherOf'(F, Ch) '←'
  'p':'parentOf'(F, Ch) '∧' 'm':'male'(F)).
```

In the above example, `'f':'fatherOf'(F, Ch) '←' 'p':'parentOf'(F, Ch) '∧' 'm':'male'(F)` is a rule expressing an object-level sentence which becomes a meta-representation (meta-term) to be manipulated at the meta-level by a meta-interpreter.

- **A meta-language for the meta-level (MML)**

Additionally, an SW ontology defines classes, properties, their relationships, as well as class-instance relations. This information is precisely *meta-information of the object level*. Here we express this information by MML in our meta-logical framework. MML includes:

Meta-constant specifying a name of an instance, a property, a class, a literal, and a namespace.

Meta-variable standing for a different meta-constant at a different time.

Meta-function symbol standing for a logical connective, e.g. `'←'`, `'∧'`; or `':'` (for namespace labeling); or a name of set operators applied on classes such as union, intersection, and complement; or a meta-predicate name being a name of a relation between entities; or being a name of characteristic of a property, which may fall into one of the following categories:

Class-class relations: subclass of, equivalent class of, disjoint with, etc.

Class-instance relations: instance of, class of, etc.

Property-property relations: sub property of, inverse of, etc.

Relations between literals and instances/classes/properties. We can see these relations as attributes of instances, of classes, or of properties, e.g. comment etc.

Characteristics of properties: symmetric, transitive, functional, etc.

Meta-term being either a meta-constant or a meta-variable or a meta-function symbol applied to a tuple of terms, e.g. `'f':'fatherOf', 'owl':'equivalentClass'(C,EC)`.

When a meta-term expresses a meta-level predicate stating a relation between entities, it has the form of `Pred(Sub,Obj)`; and when it expresses a meta-level predicate stating a characteristic of a property, it has the form of `Pred(Prop)`, where `Pred` is a meta-predicate name, `Sub`, `Obj`, and `Prop` (a property) are meta-constants or meta-variables.

The meta-term expressing a *meta-level sentence* is a term `Pred(Sub,Obj)` or `Pred(Prop)` or a logical-connective function symbol applied to the tuple of these terms. Let all meta-variables appearing in the meta-level sentence be universally quantified. One form of the sentence is a Horn-clause **meta-rule**, e.g.

```
'owl':'equivalentClass'(C, EC) ←
  'owl':'equivalentClass'(C, EC1) ∧
  'owl':'equivalentClass'(EC1, EC).
```

Meta-statement being a meta-representation of a meta-

IEEE International Conference on Computer Sciences - RIVF'06
February 12-16, 2006, Ho Chi Minh City, Vietnam

level sentence to be accessible by the meta-interpreter. It has two forms: `meta_statement(meta-level-sentence)` and `axiom(meta-level-sentence)`; the latter form represents a rule for a mathematical axiom. Here are some examples of the meta-statements:

```
meta_statement('rdfs':'subPropertyOf'(
  'f':'hasBrother','f':'hasSibling')←true).
axiom('owl':'equivalentClass'(C, EC) ←
  'owl':'equivalentClass'(C, EC1) ∧
  'owl':'equivalentClass'(EC1, EC)).
```

IV. Ontology Meta-Programs

An SW ontology is transformed into a meta-program containing a (sub-)meta-program expressed in ML, we call it "MP", and/or a (sub-)meta-program expressed in MML, we call it "MMP". Another meta-program asserting mathematical axioms expressed in MML, called "AMP", is also needed. The inference engine often requires AMP to reason with MP and MMP.

- **The meta-program for the object level (MP)**

The MP program contains information of instances and their relationships in term of meta-statements for the object level: `statement(P(S, O) ← true)`, `statement(P(S, O) ← Body)`, where `Body` expresses a single object-level predicate or a conjunction of these object-level predicates.

- **The meta-program for the meta-level (MMP)**

MMP contains description of classes, properties, their relations, class-instance relationships, and meta-rules. Here are some typical elements of an MMP program:

Some meta-statements about classes and their relationships:

```
meta_statement('rdfs':'subClassOf'(
  C, SC) ← true).
// The class C is sub-class of the class SC.
meta_statement('owl':'equivalentClass'(
  C, EC) ← true).
// Classes C & EC are equivalent.
meta_statement('owl':'disjointWith'(
  C, DC) ← true).
// Classes C & DC are disjoint.
meta_statement('owl':'unionOf'(
  C, Cs) ← true).
// The class C is union of classes in Cs.
meta_statement('owl':'intersectionOf'(
  C, Cs) ← true).
// The class C is intersection of classes in Cs.
meta_statement('owl':'complementOf'(
  C, CC) ← true).
// The class C is complement of the class CC.
meta_statement('rdf':'type'(I, C) ← true).
// The instance I is an instance of the class C.
```

Some meta-statements about properties and their relationships:

```
meta_statement('rdfs':'subPropertyOf'(
  P, SP) ← true).
// The property P is sub-property of the property SP.
meta_statement('owl':'equivalentProperty'(
  P, EP) ← true).
// Properties P & EP are equivalent.
meta_statement('owl':'symmetric'(P) ← true).
// The property P is symmetric.
meta_statement('owl':'transitive'(P)← true).
// The property P is transitive.
meta_statement('owl':'functional'(P)← true).
// The property P is functional.
meta_statement('owl':'inverseFunctional'(P)
  ← true).
// The property P is inverse functional.
meta_statement('owl':'inverseOf'(P, IP)
  ← true).
// The property P is inversion of the property IP.
meta_statement('rdfs':'domain'(P, D)← true).
// The domain of the property P is D.
meta_statement('rdfs':'range'(P, R) ← true).
// The range of the property P is R
```

- **The meta-program for the axioms (AMP)**

AMP contains axioms for classes, instances, and properties. They are expressed in the meta-rule form.

```
// the following relations of classes and properties are transitive.
axiom('owl':'equivalentClass'(C, EC) ←   (atec)
  'owl':'equivalentClass'(C, EC1) ∧
  'owl':'equivalentClass'(EC1, EC)).
axiom('rdfs':'subClassOf'(C, SC) ←       (atsc)
  'rdfs':'subClassOf'(C, SC1) ∧
  'rdfs':'subClassOf'(SC1, SC)).
axiom('owl':'equivalentProperty'(P,EP)← (atep)
  'owl':'equivalentProperty'(P, EP1) ∧
  'owl':'equivalentProperty'(EP1, EP)).
axiom('rdfs':'subPropertyOf'(P, SP) ←    (atsp)
  'rdfs':'subPropertyOf'(P, SP1)) ∧
  'rdfs':'subPropertyOf'(SP1, SP)).
axiom('owl':'sameAs'(I, SI) ←            (atsa)
  'owl':'sameAs'(I, SI1) ∧
  'owl':'sameAs'(SI1, SI)).
```

IEEE International Conference on Computer Sciences - RIVF'06
February 12-16, 2006, Ho Chi Minh City, Vietnam

```
// the following relations of classes and properties are symmetric.
axiom('owl':'equivalentClass'(C, EC) ← (asec)
  'owl':'equivalentClass'(EC, C)).
axiom('owl':'disjointWith'(C, DC) ← (asdc)
  'owl':'disjointWith'(DC, C)).
axiom('owl':'equivalentProperty'(P,EP)← (ascp)
  'owl':'equivalentProperty'(EP, P)).
axiom('owl':'inverseOf'(P, IP) ← (asip)
  'owl':'inverseOf'(IP, P)).
axiom('owl':'sameAs'(I, SI) ← (assa)
  'owl':'sameAs'(SI, I)).
axiom('owl':'differentFrom'(I, DI) ← (asdf)
  'owl':'differentFrom'(DI, I)).
...
// inheritance axiom: an instance of a subclass is also an instance
of its super class.
axiom('rdf':'type'(I, C) ← (asic)
  'rdfs':'subClassOf'(SC, C) ∧
  'rdf':'type'(I, SC)).
...
// some axioms are related to characteristics of a property
axiom(P(S, O) ← (acip)
  'owl':'inverseOf'(P, IP) ∧ IP(O, S)).
axiom(P(S, O) ← (acsp)
  'rdfs':'subPropertyOf'(SP, P) ∧ SP(S, O)).
axiom(P(S, O) ← (accp)
  'owl':'equivalentProperty'(P, EP) ∧
  EP(S, O)).
axiom(P(S, O) ← (actp)
  'owl':'transitive'(P) ∧
  P(S, O1) ∧ P(O1, O)).
axiom(P(S, O) ← (acsmp)
  'owl':'symmetric'(P) ∧ P(O, S)).
...
```

V. Semantic Web Meta-Interpreter

A meta-interpreter, yet another meta-program in our framework, takes MP, MMP—the two sub-meta-programs transformed from SW ontologies—and AMP as the inputs to manipulate and reason with. This meta-interpreter can be used to develop an intelligent agent to reason with SW ontologies. The meta-interpreter in our framework is defined by the demo predicate definition as follows.

The `demo` predicate has the form `demo(A)`; it means that the answer `A` can be infered from the meta-programs adopted by the predicate. In our framework MP, MMP, and/or AMP, are taken to be these meta-programs. Here the Vanilla meta-interpreter [11] is adapted for reasoning with meta-programs transformed from SW ontologies where we have identified three kinds of meta-level statements: (1) `statement(A '←' B)` for the object level of an ontology, (2) `meta_statement(A '←' B)` for the meta-level of an ontology, (3) `axiom(A '←' B)` for the mathematical axioms. The `demo/1` predicates are defined as follows.

```
demo(true).                                          (true)
demo(A '∧' B) ←                                      (conj)
  demo(A) ∧ demo(B).
demo(A) ←                                            (ost)
  statement(A '←' B) ∧ demo(B).
demo(A) ←                                            (mst)
  meta_statement(A '←' B) ∧ demo(B).
demo(A) ←                                            (ast)
  axiom(A '←' B) ∧ demo(B).
```

VI. Query Answering Using the Meta-Interpreter

Given an MP, an MMP, and/or an AMP to the meta-interpreter which is implemented in Prolog, we can now query the meta-interpreter to derive explicit and implicit information from an SW ontology. Next we use the family ontology taken from the library of ontologies [10] for the demonstration. Due to the size of the ontology example, only some parts of it are shown below:

- **The MMP program:**

```
meta_statement('owl':'inverseOf'(                    (1)
  'f':'hasFather','f':'hasChild') ← true).
meta_statement('rdfs':'subPropertyOf'(               (2)
  'f':'hasFather','f':'hasAncestor') ← true).
meta_statement('rdfs':'subPropertyOf'(               (3)
  'f':'hasBrother','f':'hasSibling') ← true).
meta_statement('owl':'symmetric'(                    (4)
  'f':'hasSibling') ← true).
meta_statement('owl':'transitive'(                   (5)
  'f':'hasAncestor') ← true).
...
```

- **The MP program:**

```
statement('f':'hasFather'(                           (1')
  'f':'M02', 'f':'M01') ← true).
statement('f':'hasFather'(                           (2')
  'f':'M03', 'f':'M02') ← true).
statement('f':'hasBrother'(                          (3')
  'f':'F02', 'f':'M02') ← true).
...
```

Some query answering with the meta-interpreter:

```
?- demo('owl':'inverseOf'(
        'f':'hasFather', X)).
   X = 'f':'hasChild';
   // reasoning by adopting clauses (mst), (true), (1)
```

IEEE International Conference on Computer Sciences - RIVF'06
February 12-16, 2006, Ho Chi Minh City, Vietnam

```
?- demo('owl':'inverseOf'(
        'f':'hasChild', X)).
   X = 'f':'hasFather';
```
// reasoning by adopting clauses (asip), (ast), (mst), (true), (1)

```
?- demo('f':'hasChild'('f':'M01', X)).
   X = 'f':'M02';
```
// reasoning by adopting clauses (acip), (asip), (ast), (mst), (ost), (conj), (true), (1), (1')

```
?- demo('f':'hasSibling'('f':'F02', X)).
   X = 'f':'M02';
```
// reasoning by adopting clauses (acsp), (ast), (mst), (ost), (conj), (true), (3), (3')

```
?- demo('f':'hasSibling'('f':'M02', X)).
   X = 'f':'F02';
```
// reasoning by adopting clauses (acsmp), (acsp), (ast), (mst), (ost), (conj), (true), (3), (4), (3')

```
?- demo('f':'hasAncestor'('f':'M02', X)).
   X = 'f':'M01';
```
// reasoning by adopting clauses (acsp), (ast), (mst), (ost), (conj), (true), (2), (1')

```
?- demo('f':'hasAncestor'('f':'M03', X)).
   X = 'f':'M02';
```
// reasoning by adopting the clauses (acsp), (ast), (mst), (ost), (conj), (true), (2), (2')

```
   X = 'f':'M01';
```
// reasoning by adopting the clauses (actp), (acsp), (ast), (mst),(ost), (conj), (true), (2), (5), (1'), (2')

VII. Some Implementation Issues of our Ontology Transformation

Here we show an example of an MP and an MMP that are transformed from a part of the family ontology used earlier for the query answering demonstration. The ontology example below defines a class `Man` as a subclass of a class `Human`; `M01` is a man; `M02` is another man who has `M01` as his father; `F02` is a woman and she has `M02` as her brother.

A small part of the family SW ontology:

```
<owl:Class rdf:ID='f:Man'>
  <rdfs:subClassOf rdf:resource='#Human'/>
</owl:Class>
<f:Man rdf:ID='f:M02'>
  <f:hasFather rdf:resource='#M01'/>
</f:Man>
<f:Woman rdf:ID='f:F02'>
  <f:hasBrother rdf:resource='#M02'/>
</f:Woman>
<rdfs:Property rdf:ID='f:hasFather'>
  <rdfs:subPropertyOf
    rdf:resource='#hasParent'/>
</rdfs:Property>
```

After the transformation, we get the MMP:

```
meta_statement('rdf':'type'(
  'f':'Man', 'owl':'Class') ← true).
meta_statement('rdfs':'subClassOf'(
  'f':'Man', 'f':'Human') ← true).
meta_statement('rdf':'type'(
  'f':'M02', 'f':'Man') ← true).
meta_statement('rdf':'type'(
  'f':'F02', 'f':'Woman') ← true).
meta_statement('rdf':'type'(
  'f':'hasFather','rdfs':'Property')← true).
meta_statement('rdfs':'subPropertyOf'(
  'f':'hasFather', 'f':'hasParent') ← true).
```

and the MP:

```
statement('f':'hasFather'('f':'M02',
  'f':'M01') ← true).
statement('f':'hasBrother'('f':'F02',
  'f':'M02') ← true).
```

- **Namespace and Identifier**

A fully qualified name of every web resource is used as an identifier of the corresponding entity in our system. This identifier is represented in the system by the form:

`<namespace>:<identifier>`

Here `<namespace>` is exactly the abbreviation of a namespace that is defined in the ontology source file and the `<identifier>` is the local name of the resource. By using the fully qualified names for all resources, there will not be a name conflict in the system because they are globally and uniquely defined.

VIII. Comparison with Other Related Works

Wielemaker et .al. [5] proposed a logic programming based approach to transform an SW ontology in the form of RDF and RDFS to logical triples representing the RDF triples. They also provided a Prolog library to process and query these logical triples.

Compared with our approach, firstly their approach did not make a distinction between object level and meta-level triples, i.e. their `rdf(S, P, O)` can be our `statement(P(S, O) ← true)` or `meta_statement(P(S, O) ← true)`. Secondly, their approach did not provide any inference engine but used the Prolog interpreter to be the inference engine. Thirdly, this approach provided just basic reasoning for some queries related to class and property relationships, such as, `subClassOf`, `type`, etc., but did not make use of other mathematical axioms, e.g. `inverseOf`, `symmetric`, etc. as being demonstrated below. Finally their approach did not support reasoning with OWL language.

IEEE International Conference on Computer Sciences - RIVF'06
February 12-16, 2006, Ho Chi Minh City, Vietnam

For the same family ontology used for the query answering demonstration in section VI, their transformation will give the following triples.

```
rdf('f:hasFather', 'owl:inverseOf',
  'f:hasChild').
rdf('f:hasFather', 'rdfs:subPropertyOf',
  'f:hasParent').
rdf('f:hasFather', 'rdfs:subPropertyOf',
  'f:hasAncestor').
rdf('f:hasBrother', 'rdfs:subPropertyOf',
  'f:hasSibling').
rdf('f:hasSibling', 'rdf:type',
  'owl:SymmetricProperty').
rdf('f:hasAncestor', 'rdf:type',
  'owl:TransitiveProperty').
...
rdf('f:M02', 'f:hasFather', 'f:M01').
rdf('f:M03', 'f:hasFather', 'f:M02').
rdf('f:F02', 'f:hasBrother', 'f:M02').
...
```

Then they have to use their `rdf_has/3` predicate to query from the triples by using a Prolog interpreter; the example results are:

```
Q1: ?- rdf_has('f:hasFather',
       'owl:inverseOf', X).
    X = 'f:hasChild';
Q2: ?- rdf_has('f:hasChild',
       'owl:inverseOf', X).
    No
Q3: ?- rdf_has('f:M01', 'f:hasChild', X).
    No
Q4: ?- rdf_has('f:F02', 'f:hasSibling', X).
    X = 'f:M02';
Q5: ?- rdf_has('f:M02', 'f:hasSibling', X).
    No
Q6: ?- rdf_has('f:hasAncestor', 'f:M02', X).
    X = 'f:M01';
Q7: ?- rdf_has('f:M03', 'f:hasAncestor', X).
    X = 'f:M02';
```

For `Q1`, `Q4`, and `Q6`, their approach gives the same result as ours, but for `Q2`, `Q3`, `Q5`, and `Q7` it cannot give the correct answers like what our approach does (see section VI).

IX. Conclusion

We have presented a unified framework for representing and reasoning with SW ontologies using meta-logic. With the merit of meta-logic, our system can uniformly reason in logic as well as with classes, instances, properties, and their relationships that are defined and stated in SW ontologies. Beneficially, this framework can naturally be extended to mechanize an agent communication of SW information.

References

[1] I. Horrocks, "DAML+OIL: A Reasonable Web Ontology Language", *in proc. of the EDBT 2002*, LNCS, Vol. 2287, Springer-Verlag, pp. 2-13, 2002.

[2] S. Shukla, "Semantic Web and Resource Description Framework (RDF)", *in proc. of the Workshop on Digital Libraries: Theory and Practice*, Bangalore, 2003.

[3] O. Corby, R. Dieng, and C. Hébert, "A Conceptual Graph Model for W3C Resource Description Framework", LNCS, Vol. 1867, Springer-Verlag , pp. 468-482, 2000.

[4] D. Fensel, I. Horrocks, F. van Harmelen, D. McGuinness, P. F. Patel-Schneider, "OIL: Ontology Infrastructure to Enable the Semantic Web", *IEEE Intelligent System*, 16 (2), 2001.

[5] J. Wielemaker, G. Schreiber, B. Wielinga, "Prolog-based Infrastructure for RDF: Scalability and Performance", LNCS, Vol. 2870, Springer-Verlag, pp. 644-658, 2003.

[6] J. Peer, "A Logic Programming Approach To RDF Document And Query Transformation", *in proc. of the 15th European Conference on Artificial Intelligence*, France, 2002.

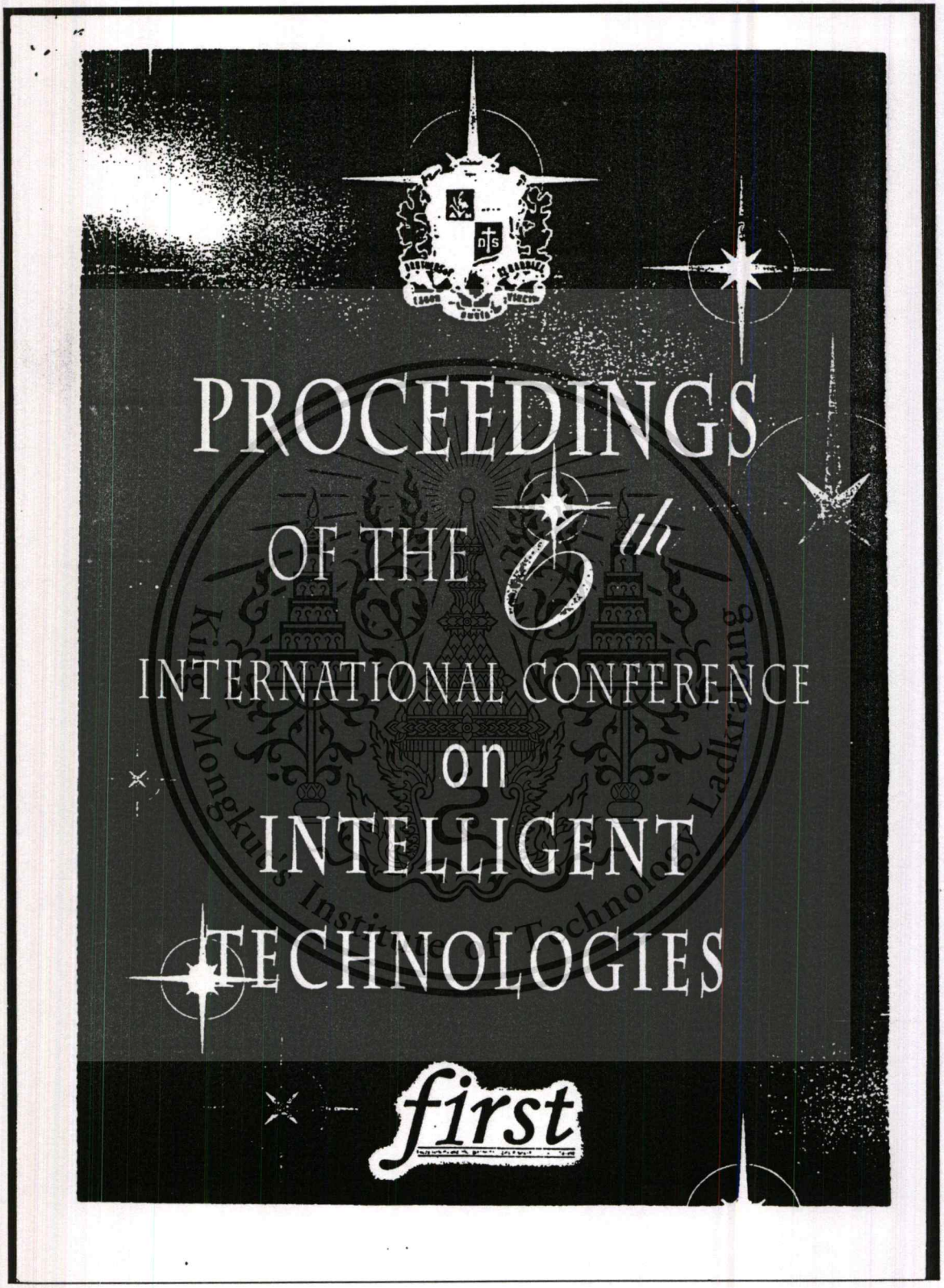
[7] N. Bassiliades, G. Antoniou, I. Vlahavas, "DR-DEVICE: A Defeasible Logic System for the Semantic Web", LNCS, Vol. 3208, Springer-Verlag, pp. 134-148, 2004.

[8] S. Koide, M. Kawamura, "SWCLOS: A Semantic Web Processor on Common Lisp Object System", *in proc. of the 3rd International Semantic Web Conference*, Japan, 2004.

[9] B. N. Grosof, I. Horrocks, "Description Logic Programs: Combining Logic Programs with Description Logic", *in proc. of the 12th International Conference on the WWW*, Hungary, 2003.

[10] The family ontology, *http://protege.stanford.edu/plugins/owl/owl-library/family.swrl.owl*

[11] R. A. Kowalski, J. S. Kim, "A Metalogic Programming Approach to Multi-agent Knowledge and Belief", *in AI and Mathematical Theory of Computation*, pp. 231-246, 1991.



Meta-reasoning with Multiple Distributed Ontologies on the Semantic Web

Visit Hirankitti
Intelligent Communication and Transportation Lab,
Department of Computer Engineering,
Faculty of Engineering, King Mongkut's Institute of Technology Ladkrabang.
Ladkrabang, Bangkok 10520, Thailand
visit@ce.kmitl.ac.th

Vuong Tran Xuan
Intelligent Communication and Transportation Lab,
Department of Computer Engineering,
Faculty of Engineering, King Mongkut's Institute of Technology Ladkrabang.
Ladkrabang, Bangkok 10520, Thailand
txvuong@yahoo.com

***Abstract*—In the Semantic Web era ontologies would be distributed, but connected and related to one another, throughout the world. We may foresee a general purpose browser-like inference engine which can reason with these different distributed ontologies and infer conclusions from them in response to the queries asked by the user. In this paper we have developed such an inference engine by adopting meta-logic. Our meta-logical system consists of several meta-programs, expressing different ontologies at the meta-level, and an inference engine in the form of a meta-interpreter defined by a demo(.) predicate. Our meta-reasoning framework can reason with multiple Semantic Web ontologies expressed in RDF, RDFS, and OWL by: (1) the different ontologies, treated as different theories, are transformed to some meta-logical statements, (2) these meta-logical statements of the different ontologies are then supplied to the meta-interpreter to draw conclusions being the answers to the queries posted by the user.**

Semantic Web, multi-ontologies, meta-reasoning, logic programming

I. Introduction

The Semantic Web (or briefly "SW"), the next generation of the web, would be the largest source of linked semantic information throughout the globe. The SW information, expressed in some XML mark-up languages, such as RDF, RDFS, DAML+OIL, OWL, etc., which describe ontologies can be understood and automatically reasoned by a computer. We use an ontology to describes a domain of discourse—many kinds of human knowledge.

At the time when the Semantic Web is replacing our current web, a large number of ontologies would be incrementally developed and added to the web; the old ontologies can be referred, re-used, or extended by the new ones. This will result in a large web of distributed but linked ontologies spanning throughout the Internet.

Essentially, there would be some demand for a browser-like inference engine which can reason with multiple distributed ontologies on the SW. This inference engine, which understood as a agent, can facilitate communication between other agents to exchange semantic information, inferred from the multiple ontologies, between each other.

There have been some research works paying attention to the problem concerning multiple ontologies on the SW. A rather traditional approach [1] used a global ontology as an intermediate component for communication among other ontologies which are all translated to the global ontology. Some approaches [2, 3, 4, 5, 6] provide mechanisms for translating a dataset from one ontology to another, merging among multi-ontologies, creating the mappings among ontologies, and querying against multi-ontologies. Due to the fact that ontology languages like DAML+OIL, OWL are closely related to Description Logic, first-order .logic, there are some logic programming approaches like [7] that transform information in ontologies to logic predicates and define some rules to map concepts and queries between these ontologies.

Most of the above approaches focus mainly on how to handle multi-ontologies but do not pay much attention to the problem of reasoning with distributed ontologies. In this work, however, we have developed a meta-logical system which can reason with multiple distributed ontologies on the web.

The rest of this paper is organized as follows. Section II establishes our position, to adopt meta-logic to reason with SW ontologies. Section III presents our meta-logical framework and section IV describes ontology meta-programs. Section V describes an SW meta-interpreter and Section VI shows how to use it to query and reason with SW ontologies. Section VII refers to some implementation issues. Section VIII compares our work with other related works. Finally, section IX concludes this paper.

II. Our Approach

In this section, we first explain the need for logical reasoning in the SW. We then address the problem regarding reasoning with multiple distributed ontologies, and after that we introduce our framework.

A. Semantic Web and Logic Programming

SW languages and logic programming languages are similar in that they are both declarative. In logic programming,

This research was supported by the Japan International Cooperation Agency (JICA) under the ASEAN University Network / Southeast Asia Engineering Education Development Network (AUN/SEED-Net) Program.

a domain of discourse is described in term of objects and their relationships together with rules which used to derive new information. In the SW, just a part of information of resources is explicitly described. Given some ontologies and rules, implicit knowledge can be derived using some inference mechanisms. Hence, a logical reasoning system should be used for this purpose.

To use logic programming for the SW, [8, 9] first transformed information of an SW ontology to a logic program and then defined rules to manipulate and reason about it. Alternatively, in this paper we propose that with logic programming we can use meta-logic and can easily build a meta-interpreter to analyze, manipulate, and reason about the logic program transformed from the ontology. Additionally, our approach can also reason with multiple distributed ontologies.

B. *The SW with Multiple Distributed Ontologies*

Similar to the current web where the web contents are distributed but linked together, the future SW would be so too, but with its contents being ontologies instead of web pages. This means that an SW browser would have to reason with these distributed ontologies.

To ease our explanation throughout this paper, from now on we shall introduce an example of multiple distributed ontologies and later use them to explain with our framework throughout.

A simple and familiar example is a collection of family ontologies. Here we took a family example from a library of ontologies [10]. This family ontology contains a rather complete set of definitions for describing family relations. For the other family ontologies, they describe information about some European royal families; we took them from [11] and [12]. These two genealogy ontologies contain a large amount of family information but without definitions of family relations. The three family ontologies represent an example of some distributed ontologies (on the SW) to be reasoned by our meta-logical system in which relations defined in the first ontology are used to reason and derive relationships between and among members of European royal families in the two other ontologies.

C. *Reasoning with Multiple Distributed Ontologies*

Our meta-logical system consists of two parts: meta-programs for individual ontologies and a meta-interpreter. Each of the meta-programs contains a meta-level description of each ontology taken from the SW. That is, the ontology expressed in its native form, e.g. in RDF, RDFS, and OWL languages, is transformed to a meta-logical form. Some elements in one ontology may be related to some elements in another ontology. The meta-interpreter is the inference engine of the system which is used to infer explicit and implicit information from each ontology.

The way the system works can be explained as follows. Initially the user requests the system to obtain an SW ontology from the web; this ontology is then transformed to a meta-program. Now the user can ask some information from the system; the user query will be posted to the interpreter to request for an answer. The interpreter then reasons with the meta-program to infer an answer. If some elements in this ontology are related to some elements of another ontology, the interpreter will then retrieve that ontology from the web and transform it to a meta-program and reason with it; this scenario may repeat itself with another ontology until the interpreter can finally reach any conclusion for the answer. A diagram illustrating our system is given in Fig. 1.

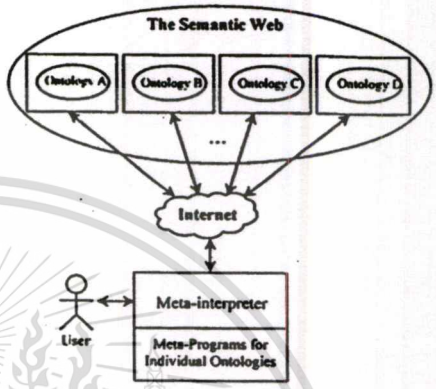


Figure 1. Our multi-ontologies reasoning system

In the next section the framework of our meta-logical system is presented. This framework is formulated in meta-logic.

III. OUR META-LOGICAL FRAMEWORK

Our framework forms a logical system consisting of logical sentences and an inference engine. The former is in the form of a meta-program representing an SW ontology; the latter is a meta-interpreter, in the form of a demo (meta-)program, which infers implicit information and draws conclusions from the former. The interpreter also communicates to the Internet to obtain SW ontologies, communicates with the user (and other agents) to get SW information, draws inferences for the user, and traverses a link to an SW or web resource like a web browser. Our framework presented in this paper is developed with the purpose to facilitate the system to reason with multiple distributed ontologies.

Our meta-logical system for SW can be illustrated simply as in Fig. 2. Firstly, an SW ontology is retrieved via the Internet and is translated to a meta-program. The user can query the interpreter which in turn derives answers from the transformed meta-program. If the answers contain links, it can traverse the links to get the corresponding web resources.

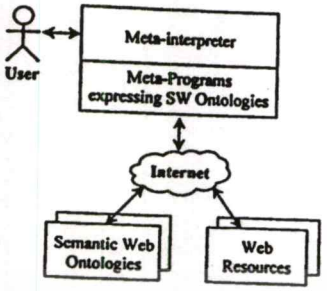


Figure 2. Our meta-logical system for the SW

To describe our framework, in the following we first develop two meta-languages of an SW ontology by mapping the language elements of RDF, RDFS, and OWL into meta-language terms, and then give an example of some parts of the three meta-programs. Finally, we give details of our meta-interpreter together with its demonstration on query answering.

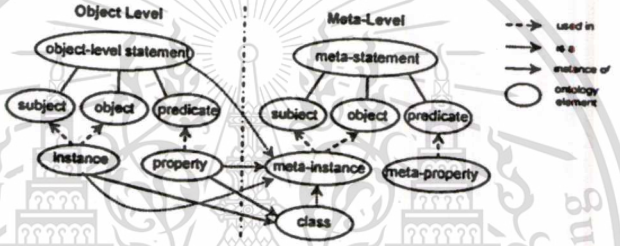


Figure 3. The elements of an SW ontology at the object level and the meta-level

Language Elements of an SW Ontology

The language elements of an SW ontology are classes, properties, class instances, and relationships between and among them described in the object level and the meta-level as illustrated in Fig. 3.

At the object level, an instance can be an individual or a literal in a domain, e.g. `'john'`, and a property is a relationship between individuals or an individual's attribute, e.g. `'hasFather'`, `'name'`. At the meta-level, a meta-instance can be an individual, a property, a class, and an object-level statement; and a meta-property is a property to describe a relationship between and among meta-instances, or a meta-instance's attribute, e.g. `'subClassOf'`, `'equivalentProperty'`, etc. Due to the use of namespace to make a name unique in an SW ontology, henceforth a qualified name in an SW ontology will be written as `<namespace>:<name>`, for example, `'f':'john'`, `'owl':'equivalentProperty'`, etc. Next we define meta-logical terms to express these elements of the ontology languages at the object level and the meta-level.

B. Meta-languages of an SW Ontology

In our meta-logical framework we distinguish between the object language and the meta-language of an SW ontology. The object language specifies objects and their relationships in the real world. The meta-language describes the syntactic form of the object language. For example, in the real world John is the father of Tom, but at the meta-language `'John'` and `'Tom'` are names representing nouns in the sentence.

More precisely, we break the meta-language elements into 2 parts: one discusses about objects and their relationships, we call it "meta-language for the object level (ML)"; the other, called "meta-language for the meta-level (MML)", discusses about classes, class instances, properties, and their relationships.

- **A meta-language for the object level (ML)**

Objects and their relationships at the object level are specified in the ontology and this information is expressed at the meta-level by meta-constants, meta-variables, meta-function symbols, meta-terms, and meta-statements as follows.

Meta-constant specifies a name of an object and a literal, e.g. `'john'`, including a reference, e.g. a namespace, an ontology name. The latter is a meta-constant from MML (see next). This means that ML and MML are not entirely separated.

Meta-variable stands for a different meta-constant at a different time, e.g. `Person`.

Meta-function symbol stands for the name of a relation between objects, or of an object's property—i.e. an object-level predicate name, such as `'fatherOf'`, `'name'`. The meta-function symbol also stands for other meta-level function symbol, e.g. `'←'`, `'^'`, `':'`, `'#'`. The meta-function symbol can also be a term of the form `<ontology_name>#<namespace>:<object-level predicate name>` where `'#'` and `':'` are meta-function symbols, e.g. `'onto'#'f':'fatherOf'`, and `<ontology_name>` and `<namespace>` are meta-constants or meta-variables.

Meta-term is either a meta-constant or a meta-variable or a meta-function symbol applied to a tuple of terms, e.g. `'f':'M1'`, `'onto'#'f':'fatherOf'`.

To express an object-level predicate, it has the form: `P(S, O)` where `P` is an object-level predicate name, `S` and `O` are meta-constants or meta-variables (we presume all meta-variables appearing in the tuple are universally quantified), e.g. `'f':'fatherOf'('f':'M2', 'f':'M1')`. To express a provability predicate, it has the form: `demo(T, P(S, O))`, e.g. `demo(o, 'f':'fatherOf'('f':'M2', 'f':'M1'))`.

The meta-term expressing an object-level sentence (sometimes with some provability) is a term `P(S, O)` or `demo(T, P(S, O))` or a logical-connective function symbol applied to the tuple of these terms. One form of it is a Horn-clause, e.g. `'f':'fatherOf'(F, Ch) ← demo(T, 'p':'parentOf'(F,Ch)) ∧ demo(T, 'm':'male'(F))`.

Meta-statement for the object level reflects an object-level sentence to its existence at the meta-level. It has the form: `statement(T, object-level-sentence)`, where `T` is an ontology name, e.g.

```
statement(T, 'f':'fatherOf'(F, Ch) ←
  demo(T, 'p':'parentOf'(F, Ch)) ∧
  demo(T, 'm':'male'(F))).
```

- **A meta-language for the meta-level (MML)**

Additionally, an SW ontology defines classes, properties, and their relationships, and it also describes class-instance relations. This information is precisely meta-information of the object level. Here we express this information by MML in our meta-logical framework. MML includes:

Meta-constant specifying a name of an agent, a resource location, a namespace, an ontology, an instance, a property, a class, and a literal.

Meta-variable standing for a different meta-constant at a different time.

Meta-function symbol naming a function.

Meta-term is either a meta-constant or a meta-variable or a meta-function symbol applied to a tuple of terms.

Meta-predicate name naming a relation between entities, or a characteristic of a property, which may fall into one of the following categories:

Class-class relations: subclass of, equivalent class of, disjoint with, etc.

Class-instance relations: instance of, class of, etc.

Property-property relations: subproperty of, inverse of, etc.

Relations between literals and instances/classes/properties. We can see these relations as attributes of instances, classes, or properties, e.g. comment, label, etc.

Characteristics of properties: such as functional, some mathematical properties, e.g. transitive, symmetric, etc.

Meta-predicate expressing a relation between entities having the form of `Pred(Sub, Obj)`, or a characteristic of a property having the form of `Pred(Prop)`, where `Pred` is a meta-predicate name, `Sub`, `Obj`, and `Prop` (a property) are meta-terms, e.g. `'owl':'inverseOf'('f':'fatherOf', 'f':'childOf')`. Let all the meta-variables appearing in a meta-predicate be universally quantified.

Meta-operator expressing a set operation between classes such as union.

Meta-statement being a meta-predicate or meta-predicates connected by logical connectives. One form of the meta-statement is a Horn-clause meta-rule. Here are some examples of the meta-rules:

```
meta_statement(o, 'owl':'inverseOf'(
  'f':'fatherOf', 'f':'childOf') ← true)
axiom(t, 'owl':'equivalentClass'(C, EC) ←
  'owl':'equivalentClass'(C, EC1) ∧
  'owl':'equivalentClass'(EC1, EC)).
```

The second example represents a rule for a mathematical property—equivalence; from now we shall call a meta-statement which represents a mathematical property as an 'axiom' instead of a 'meta-statement'.

IV. Ontology Meta-Programs

Each SW ontology is transformed to a meta-program containing a (sub-)meta-program expressed in ML, we call it "MP", and/or a (sub-) meta-program expressed in MML, we call it "MMP". Another meta-program, which is also needed for the inferential purpose in our framework, expresses some mathematical axioms; we shall call it "AMP". The inference engine requires AMP in order to support reasoning with MMP.

- **The meta-program for the object level (MP)**

The MP program contains information of instances and their relationships in terms of meta-statements for the object level: `statement(T, P(S, O) ← true)` and `statement(T, P(S, O) ← Body)`, where `Body` is a conjunction of meta-predicates for the object level; the latter is the rule form for the object level. It is important to note that in order to specify that the statements `P(S, O) ← true` and `P(S, O) ← Body` belong to a particular ontology `T` (stands for a theory name), we shall put `T` as the first argument in `statement()`; and this form of ontology labeling will be used henceforth.

- **The meta-program for the meta-level (MMP)**

MMP contains description of classes, properties, their relations, class-instance relationships in term of meta-rules. Some typical elements of an MMP program are depicted in Fig. 4.

- **The meta-program for the axioms (AMP)**

AMP contains axioms for classes and properties. The axioms are expressed in the meta-rule form, e.g. the axiom about the equivalence of classes is defined as follows.

```
axiom(T, 'owl':'equivalentClass'(C, EC) ←
  'owl':'equivalentClass'(EC, C)).
axiom(T, 'owl':'equivalentClass'(C, EC) ←
  'owl':'equivalentClass'(C, EC1) ∧
  'owl':'equivalentClass'(EC1, EC)).
```

Some typical axioms of an AMP program are depicted in Fig. 5.

```
// some meta statements are related to classes
meta_statement(T, 'rdfs':'subClassOf'(C, SC) ← true).          // C is sub-class of SC.
meta_statement(T, 'owl':'equivalentClass'(C, EC) ← true).      // C & EC are equivalent.
meta_statement(T, 'owl':'disjointWith'(C, DC) ← true).         // C & DC are disjoint.
meta_statement(T, 'owl':'unionOf'(C, Cs) ← true).              // C is union of classes in Cs.
meta_statement(T, 'owl':'intersectionOf'(C, Cs) ← true).       // C is intersection of classes in Cs
meta_statement(T, 'owl':'complementOf'(C, CC) ← true)          // C is complement of CC.
meta_statement(T, 'rdf':'type'(I, C) ← true).                  // I is an instance of C.
-

// some meta statements are related to properties
meta_statement(T, 'rdfs':'subPropertyOf'(P, SP) ← true).       // P is sub-property of SP.
meta_statement(T, 'owl':'equivalentProperty'(P,EP) ← true).    // P & EP are equivalent.
meta_statement(T, 'owl':'symmetric'(P) ← true).                // P is symmetric.
meta_statement(T, 'owl':'transitive'(P) ← true).               // P is transitive.
meta_statement(T, 'owl':'functional'(P) ← true).               // P is functional.
meta_statement(T, 'owl':'inverseFunctional'(P) ← true).        // P is inverse functional.
meta_statement(T, 'owl':'inverseOf'(P, IP) ← true).            // P is inversion of IP.
meta_statement(T, 'rdfs':'domain'(P, D) ← true).               // The domain of the P is D.
meta_statement(T, 'rdfs':'range'(P, R) ← true).                // The range of the P is R.
-
```

Figure 4. The MMP program

```
// the following relations of classes and properties are transitive.
axiom(T, 'owl':'equivalentClass'(C, EC) ←
  'owl':'equivalentClass'(C, EC1) ∧ 'owl':'equivalentClass'(EC1, EC)).                (atec)
axiom(T, 'rdfs':'subClassOf'(C, SC) ←
  'rdfs':'subClassOf'(C, SC1) ∧ 'rdfs':'subClassOf'(SC1, SC)).                        (atsc)
axiom(T, 'owl':'equivalentProperty'(P, EP) ←
  'owl':'equivalentProperty'(P, EP1) ∧ 'owl':'equivalentProperty'(EP1, EP)).          (atep)
axiom(T, 'rdfs':'subPropertyOf'(P, SP) ←
  'rdfs':'subPropertyOf'(P, SP1)) ∧ 'rdfs':'subPropertyOf'(SP1, SP)).                 (atsp)
axiom(T, 'owl':'sameAs'(I, SI) ←
  'owl':'sameAs'(I, SI1) ∧ 'owl':'sameAs'(SI1, SI)).                                  (atsa)
-

// the following relations of classes and properties are symmetric.
axiom(T, 'owl':'equivalentClass'(C, EC) ← 'owl':'equivalentClass'(EC, C)).            (asec)
axiom(T, 'owl':'disjointWith'(C, DC) ← 'owl':'disjointWith'(DC, C)).                  (asdc)
axiom(T, 'owl':'equivalentProperty'(P, EP) ← 'owl':'equivalentProperty'(EP, P)).      (asep)
axiom(T, 'owl':'inverseOf'(P, IP) ← 'owl':'inverseOf'(IP, P)).                        (asip)
axiom(T, 'owl':'sameAs'(I, SI) ← 'owl':'sameAs'(SI, I)).                              (assa)
axiom(T, 'owl':'differentFrom'(I, DI) ← 'owl':'differentFrom'(DI, I)).                (asdf)
-

// an instance of a subclass is also an instance of its super class.
axiom(T, 'rdf':'type'(I, C) ← 'rdfs':'subClassOf'(SC, C) ∧ 'rdf':'type'(I, SC)).      (asic)
-

// some axioms express characteristics of properties
axiom(T, P(S, O) ← 'owl':'inverseOf'(P, IP) ∧ IP(O, S)).                              (acip)
axiom(T, P(S, O) ← 'rdfs':'subPropertyOf'(SP, P) ∧ SP(S, O)).                         (acsp)
axiom(T, P(S, O) ← 'owl':'equivalentProperty'(P, EP) ∧ EP(S, O)).                     (acep)
axiom(T, P(S, O) ← 'owl':'transitive'(P) ∧ P(S, O1) ∧ P(O1, O)).                      (actp)
axiom(T, P(S, O) ← 'owl':'symmetric'(P) ∧ P(O, S)).                                   (acsmp)
-
```

Figure 5. The AMP program

V. THE SW META-INTERPRETER (SWMI) FOR MULTI-ONTOLOGIES REASONING

A meta-interpreter, yet another meta-program in our framework, takes MPs, MMPs, and AMPs—the two sub-meta-programs transformed from the SW ontologies and the axiom meta-program—as the inputs to process and reason with. This interpreter is an extension of the Vanilla meta-interpreter proposed in [13].

The definition of `demo/1` in [13] has been extended and simplified to `demo/2`. So `demo(T, A)` means the answer `A` can be inferred from the theory `T`. This interpreter can be understood as a kind of the Vanilla meta-interpreter [13]. Here the Vanilla meta-interpreter is adapted to use for reasoning with multiple ontologies on the SW where we have identified three kinds of meta-level statements, (1) meta-statements for the object level represented by `statement(T, A ← B)`, (2) meta-statements, for the meta-level of an ontology, represented by `meta_statement(T, A ← B)`, and (3) the mathematical axioms represented by `axiom(T, A ← B)`. The definition of `demo/2` is given as follows.

```
demo(empty, true).                                   (true)
demo(T, demo(T, A)) ← demo(T, A).                    (ref)
demo(T1 ∪ T2, A) ←                                   (ost)
  statement(T1, A '←' B) ∧
  demo(T2, B).
demo(T1 ∪ T2, A) ←                                   (mst)
  meta_statement(T1, A '←' B) ∧
  demo(T2, B).
demo(T1 ∪ T2, A) ←                                   (ast)
  axiom(T1, A '←' B) ∧
  demo(T2, B).
demo(T1 ∪ T2, A '∧' B) ←                             (conj)
  demo(T1, A) ∧ demo(T2, B).
```

The clauses `(true)`, `(ost)`, and `(conj)` form the Vanilla meta-interpreter. The clause `(ref)` is rather interesting. This means that when the meta-interpreter tries to prove `demo(T, demo(T, A))`, it will prove `demo(T, A)` by a reflection.

For distributed ontologies, some ontologies may be referred to in other ontologies. In this case while `demo()` is reasoning with an ontologies to derive an answer, this may require the `demo()` to reason with another unavailable ontology as well. Here we should add another `demo()` clause to allow the demo predicate to retrieve that ontology from its location on the web, then transform it to MP and MMP, and can finally reason with it to complete all the inference steps so that it can derive the answer:

```
demo(T, demo(T, A)) ←                                (retr)
  unavailable(T) ∧
  retrieve(O, location(T)) ∧
  transform(O, P) ∧
  demo(P, A).
```

With this new clause `(retr)`, `demo()` can now work analogously to a web browser.

Additionally, suppose that each SW server storing an ontology is equipped with this `demo()` inference engine. Now for the demo (at the client) to derive an answer from an unavailable ontology; this can be done easily by that the `demo()` at the client posts the query (for an unavailable ontology) directly to the SW server which has that ontology to answer the query. For this to be done, we may add another `demo()` clause:

```
demo(T, demo(T, A)) ←                                (comm)
  unavailable(T) ∧
  find_server(S, return_server(T)) ∧
  get_connected(S) ∧
  ask_server(S, demo(T, A)) ∧
  disconnect(S).
```

This clause makes `demo()` function like a service provider in the current web service technology.

Given all above demo clauses, an answer `A` can be inferred from `demo()` in different ways: firstly `A` may be inferred using some statements in one or many MPs, and/or using meta-statements in MMPs, and/or using axioms in AMPs. The inference may require `demo()` to retrieve some ontologies from different sources on the SW or to post `demo(A)` to other remote SW servers to obtain the answer.

VI. QUERY ANSWERING USING SWMI

Given an MP, an MMP, an AMP to SWMI, we can now query SWMI to obtain explicit and derived information from the distributed SW ontologies.

In the following we use the three genealogy ontologies taken from [10, 11, 12] as an example of multiple distributed ontologies for the demonstration. Due to the size of the three ontologies, only some parts of them are shown in Fig. 6.

In this figure, the CMP has meta-statements defining some rules for mapping the relationships among three ontologies, e.g., the first meta-statement cf1 states that the property `'b':'place'` of the bbn_ged ontology [11] and the property `'d':'location'` of the drc_ged ontology [12] are equivalent; the second meta-statement is a rule defining the relation `'f':'hasFather'` (from the family ontology) based on the relations `'b':'sex'`, `'b':'spouseIn'`, and `'b':'childIn'` from the bbn_ged ontology. FMP and CMP are both required, as demonstrated in Fig. 7, for supporting the meta-interpreter to perform reasoning in order to derive many implicit relations that are not defined explicitly in these ontologies.

The FMP contains meta-statements about the relationships (*inversion*, *sub-property*) and the characteristic (*symmetric*) of the properties `'f':'hasFather'`, `'f':'hasChild'`, `'f':'hasBrother'`, and `'f':'hasSibling'`.

BMP and DMP give information of some family relations using different terminology but these relations are somehow related to the family relations in FMP.

The demonstration of query answering is shown in Fig. 7. According to the demonstration, the queries q1 and q2 are related to just the family ontology [10]. That is, to answer q1,

the interpreter can get the answer directly form FMP, but to answer q2 the interpreter has to reason with the inversive characteristic of the 'f':'hasFather' property, the 'f':'hasChild' property, and some statements in AMP.

For queries q3, q4, and q5, some relationship definitions of the family ontology, i.e. the definitions of 'f':'hasFather' and 'f':'hasSibling', are used in the inferential process for answering the queries about the relationships of the individuals in the bbn_ged ontology; this is because the definitions of these relations and many others are not given in the bbn_ged ontology. To answer such queries, the interpreter has to perform reasoning with different ontologies. For the last two queries, q6 and q7, this demonstrates how to answer the queries about some relationships of individuals in the bbn_ged ontology by using some equivalent relations in the drc_ged ontology. The information telling what assumptions are used in the reasoning steps performed by the interpreter is also noted below the answer of each query as seen in Fig. 7.

CMP: *Meta-program for the cross-referenced ontology*

```
meta statement(bbn u drc,
   'owl':'equivalentProperty'('bbn'#'b':'place', 'drc'#'d':'location') ← true).                (cf1)
meta statement(fo u T, 'fo'#'f':'hasFather'(Child, Father) ←
   demo(T, 'bbn'#'b':'sex'(Father, 'M')) ∧                                                     (cf2)
   demo(T, 'bbn'#'b':'spouseIn'(Father, Family)) ∧
   demo(T, 'bbn'#'b':'childIn'(Child, Family))).
meta statement(fo u T, 'fo'#'f':'hasBrother'(Person, Brother) ←                                (cf3)
   demo(T, 'bbn'#'b':'sex'(Brother, 'M')) ∧
   demo(T, 'bbn'#'b':'childIn'(Brother, Family)) ∧
   demo(T, 'bbn'#'b':'childIn'(Person, Family))).
meta statement(drc u T, 'drc'#'d':'husband'(Husband, Family) ←                                 (cf4)
   demo(T, 'bbn'#'b':'sex'(Husband, 'M')) ∧
   demo(T, 'bbn'#'b':'spouseIn'(Husband, Family))).
–
```

BMP: *Meta-program for the Bbn_ged ontology*

```
statement(bbn, 'bbn'#'b':'marriage'('bbn'#'b':'@F01@','bbn'#'b':'event01') ← true).          (b1)
statement(bbn, 'bbn'#'b':'place'('bbn'#'b':'event01', 'Grafton Regis') ← true).              (b2)
statement(bbn, 'bbn'#'b':'sex'('bbn'#'b':'@I01@', 'M') ← true).                              (b3)
statement(bbn, 'bbn'#'b':'spouseIn'('bbn'#'b':'@I01@', 'bbn'#'b':'@F01@') ← true).           (b4)
statement(bbn, 'bbn'#'b':'sex'('bbn'#'b':'@I02@', 'M') ← true).                              (b5)
statement(bbn, 'bbn'#'b':'childIn'('bbn'#'b':'@I02@', 'bbn'#'b':'@F01@') ← true).           (b6)
statement(bbn, 'bbn'#'b':'sex'('bbn'#'b':'@I03@', 'F') ← true).                              (b7)
statement(bbn, 'bbn'#'b':'childIn'('bbn'#'b':'@I03@', 'bbn'#'b':'@F01@') ← true).           (b8)
–
```

FMP: *Meta-program for the family ontology*

```
meta statement(fo, 'owl':'inverseOf'('fo'#'f':'hasFather', 'fo'#'f':'hasChild') ← true).    (f1)
meta statement(fo, 'rdfs':'subPropertyOf'('fo'#'f':'hasBrother',                             (f2)
   'fo'#'f':'hasSibling') ← true).
meta statement(fo, 'owl':'symmetric'('fo'#'f':'hasSibling') ← true).                         (f3)
–
```

DMP: *Meta-program for the Drc_ged ontology*

```
statement(drc, 'drc'#'d':'location'('drc'#'d':'event1138', 'Dartford') ← true).             (d1)
–
```

Figure 6. The MMP and MP programs for the demonstration

```
?- demo(T, 'owl':inverseOf('fo'#'f':'hasFather', X)).                    (q1)
   T = fo, X = 'fo'#'f':'hasChild';
   // reasoning by adopting clauses (true), (inst), and (f1)

?- demo(T, 'owl':inverseOf('fo'#'f':'hasChild', X)).                     (q2)
   T = fo, X = 'fo'#'f':'hasFather';
   // reasoning by adopting clauses (true), (inst), (ast), (asip), and (f1)

?- demo(T, 'fo'#'f':'hasFather'('bbn'#'b':'eI02e', X)).                  (q3)
   T = fo ∪ bbn, X = 'bbn'#'b':'eI01e';
   // reasoning by adopting clauses (true), (ref), (ost), (inst), (ast), (conj), (cf2), (b3), (b4), and (b6)

?- demo(T, 'fo'#'f':'hasSibling'('bbn'#'b':'eI02e', X)).                 (q4)
   T = fo ∪ bbn, X = 'bbn'#'b':'eI03e';
   // reasoning by adopting clauses (true), (ref), (ost), (inst), (ast), (conj), (acsp), (f2), (f3), (b5), (b6), and (b8)

?- demo(T, 'fo'#'f':'hasSibling'('bbn'#'b':'eI03e', X)).                 (q5)
   T = fo ∪ bbn, X = 'bbn'#'b':'eI02e';
   // reasoning by adopting clauses (true), (ref), (ost), (inst), (ast), (conj), (acsp), (acsym), (f2), (f3), (cf3), (b5), (b6), and (b8)

?- demo(T, 'drc'#'d':'location'('bbn'#'b':'event01', X)).                (q6)
   T = drc ∪ bbn, X = 'Grafton Regis';
   // reasoning by adopting clauses (true), (ref), (ost), (inst), (ast), (conj), (asep), (acep), (cf1), and (b2)

?- demo(T, 'drc'#'d':'husband'('bbn'#'b':'eI01e', X)).                   (q7)
   T = drc ∪ bbn, X = 'bbn'#'b':'eF01e';
   // reasoning by adopting clauses (true), (ref), (ost), (inst), (ast), (conj), (cf4), (b3), and (b4)
```

Figure 7. Query answering with the SWMI

VII. Implementation Issues

Before reasoning with SW ontologies, the ontologies have to be transformed to MPs and MMPs. The RDF statements expressing the definitions of classes, properties, their relationships, and class-instance relationships are translated to meta-statements that form the MMP, while the RDF statements expressing information and relationships of instances are translated to statements that form the MP. The ontology name is added to the first argument of every meta-statement as well as to the name of every resource. After transforming ontologies to the meta-programs, we can assert them to the knowledge base of the meta-interpreter to perform reasoning.

Here we show an example of an MP and an MMP that are transformed from a part of the bbn_ged genealogy ontology, named as 'bbn' in our system. The ontology example below defines a family '**eF01e**' as an instance of the class Family and having the marriage event '**event01**'. It also defines a man '**eI01e**' as an instance of the class '**Individual**' and as a spouse in the family '**eF01e**'. The '**b**' is the abbreviation of the namespace of the bbn_ged ontology.

```
<b:Family rdf:ID='b:eF01e'>
  <b:marriage rdf:resource='b:event01'>
</b:Family>

<b:Individual rdf:ID='b:eI01e'>
  <b:sex>M</b:sex>
  <b:spouseIn rdf:resource='b:eF01e'>
</b:Individual>
```

After the transformation, we get the MMP:

```
meta_statement(bbn, 'rdf':'type'(
  'bbn'#'b':'eF01e', 'bbn'#'b':'Family')).

meta_statement(bbn, 'rdf':'type'(
  'bbn'#'b':'eI01e',
  'bbn'#'b':'Individual')).
```

and the MP:

```
statement(bbn, 'bbn'#'b':'marriage'(
  'bbn'#'b':'eF01e', 'bbn'#'b':'event01')).

statement(bbn, 'bbn'#'b':'sex'(
  'bbn'#'b':'eI01e', 'M')).

statement(bbn, 'bbn'#'b':'spouseIn'(
  'bbn'#'b':'eI01e', 'bbn'#'b':'eF01e')).
```

- **Namespace and Object Identifier**

A fully qualified name of every web resource and an ontology name are combined to form an identifier of the corresponding entity in our system. This identifier is represented in the system by the form:

`<ontology_name>#<namespace>:<identifier>`

Here `<ontology_name>` is the name of the ontology in which the resource is defined, `<namespace>` is exactly the abbreviation of a namespace that is defined in the ontology source file and `<identifier>` is the local name of the resource. By using the fully qualified names of resources, there will not be a name conflict in the system because they are defined globally and uniquely. The `<ontology_name>` is treated as a name for the theory `T` in `demo(T, A)` or `statement(T, _)` in our framework.

VIII. Comparison with Other Related Works

There are some works on SW ontologies that also related to logic programming. Here we do the comparison between our approach and some of them.

Wielemaker et .al. [8] proposed a logic programming approach to transform an SW ontology in the form of RDF and RDFS to some logical triples representing the RDF triples. They also provided a Prolog library to process and query these logical triples at the object level using a built-in (Prolog) interpreter. Compared with our approach, firstly their approach did not make a distinction between object level and meta-level triples, i.e. their `rdf(S, P, O)` can be our `statement(T, P(S, O))` and `meta_statement(T, P(S, O))`; and they did not support for reasoning with multiple ontologies. Secondly, their approach did not provide any inference engine but used the Prolog interpreter to do that job. Thirdly, their approach provided just a simple way of reasoning for some queries related to class and property relationships, such as, `subClassOf`, `instanceOf`, etc., but not for other mathematical properties, e.g. `inverseOf`, `symmetric`, etc in OWL language.

Peer [9] also proposed a logic programming approach to work with multiple ontologies where RDF statements are translated to triples, and queries are represented in term of Horn clauses. This approach also proposed a mechanism for translating a set of RDF data and RDF queries from one RDF document to another by defining a mapping schema. Similarly Peer did all that at the object level. It is more sensible to reason with different theories at the meta-level.

IX. Conclusion

We have proposed a meta-logical framework for a logical system which can reason with multiple distributed ontologies on the SW. This system consists of many meta-programs transformed from many distributed ontologies and an inference engine defined by a `demo` predicate. This system can naturally be extended to mechanize a simple multi-agent communication for multiple distributed ontologies.

References

[1] A. Fernandez, A. M. de C. Moura, and F. Porto, "An Ontology-Based approach for Organizing, Sharing, and Querying Knowledge Objects on the Web", in proc. of the 14th International Workshop on Database and Expert Systems Applications, pp. 604-609, 2003.

[2] D. Dou, D. McDermott, and P. Qi, "Ontology Translation on the Semantic Web", in proc. of the International Conference on Ontologies, Databases and Applications of Semantics, November, 2003.

[3] L. Mota and L. Botelho, "OWL Ontology Translation for the Semantic Web", in proc. of the International Workshop on the Semantic Computing Initiative, May, 2005.

[4] B. Xu, P. Wang, J. Lu, Y. Li, and D. Kang, "Theory and Semantic Refinement of Bridge Ontology Based on Multi-ontologies", in proc. of the 16th International Conference on Tools with Artificial Intelligence, pp. 442-449, November, 2004.

[5] S. Bowers and L. Delcambre, "Representing and Transforming Model-Based Information", in proc. of the Semantic Web Workshop, pp. 5-12, September, 2000.

[6] G. Stumme and A. Maedche, "Ontology Merging for Federated Ontologies on the Semantic Web", in proc. of the International Workshop for Foundations of Models for Information Integration, pp. 413-418, 2001.

[7] B. N. Grosof and I. Horrocks, "Description Logic Programs: Combining Logic Programs with Description Logic", in proc. of the 12th International Conference on the World Wide Web, May, 2003.

[8] J. Wielemaker, G. Schreiber, and B. Wielinga, "Prolog-based Infrastructure for RDF: Scalability and Performance", LNCS, Volume (2870), Springer-Verlag, pp. 644-658, 2003.

[9] J. Peer, "A Logic Programming Approach To RDF Document And Query Transformation", in proc. of the 15th European Conference on Artificial Intelligence, July, 2002.

[10] The family ontology, http://protege.stanford.edu/plugins/owl/owl-library/family.swrl.owl.

[11] The bbn_ged geneology ontology, http://www.daml.org/2001/01/gedcom/royal92.

[12] The drc_ged geneology ontology, http://orlando.drc.com/daml/ontology/geneology/3.1/geneology-ont.

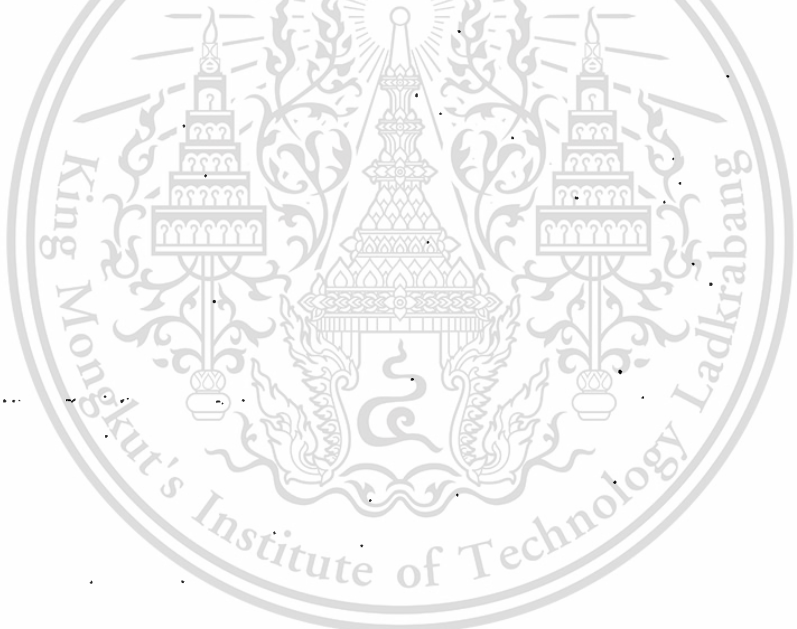
[13] R. A. Kowalski and J. S. Kim, "A Metalogic Programming approach to Multi-agent Knowledge and Belief", in Artificial intelligence and mathematical theory of computation, pp. 231-246, 1991.

ISSN 1345-0980

INAP 2005

The Proceedings of the 16th International Conference
on Applications of Declarative Programming
and Knowledge Management

22-24, October 2005
Waseda University, Fukuoka, Japan



INAP Organizing Committee

A Meta-logical Approach for Multi-agent Communication of the Semantic Web Information

Visit Hirankitti and Vuong Tran Xuan

Intelligent Communication and Transportation Laboratory,
Department of Computer Engineering, Faculty of Engineering,
King Mongkut's Institute of Technology Ladkrabang, Bangkok, Thailand
visit@ce.kmitl.ac.th, tvvuong@yahoo.com

Abstract. The success of the semantic web would be determined by how easy and uniform to access to and exchange of the semantic information among computers. In this paper we have developed a framework of multi-agent communication of the Semantic Web information. The agent and the communication between agents are characterized in meta-logic. One single agent, understood as a meta-logical system, adopts a demo(.) predicate as its inference engine and meta-programs—transformed from some Semantic Web ontologies—as its assumptions. Such an agent can reason with its assumptions as well as other agent's assumptions. With this ability, when several agents are created by using this framework, the community of these agents can uniformly communicate the Semantic Web information between each other on the Internet.

1 Introduction

The success of the Semantic Web (or briefly "SW") would be similar to that of the web in that its success is mainly due to the easy and uniform way to access to and exchange of information among the computers on the Internet. Due to this significance, in this paper we shall propose a model of communication of semantic information among multiple agents using meta-logic.

Some previous works on an agent system related to SW are: Zou et. al. [4] used SW languages, as the languages for expressing agent's messages and knowledge base, to specify and publish common ontologies; [5] presented a multi-agent based scheduling application in which data sources are described by SW languages and encapsulated in the agents. In [6], an agent is built to perform scheduling with distributed ontologies about events, e.g. conferences, classes, published on the SW.

Those approaches are mainly related to applying the SW technology in a multi-agent system. However in SW there exist a large number of available distributed but linked ontologies, hence based on our previous work [2] in this paper we are concerned with multi-agent communication and reasoning with distributed ontologies.

The rest of this paper is organized as follows. Next we give an overview of our framework. Section 3 presents our meta-representation of SW ontologies and section 4 describes our single agent architecture. Section 5 describes the meta-interpreter, the

agent's inference engine, and section 6 introduces multi-agent communication. Section 7 shows how to query and reason with SW ontologies by multi-agent communication. Finally, we discuss about other related works and conclude this paper.

2 Our Framework

In our previous work [2] we applied meta-logic to develop a meta-logical system behaving as an agent, and in this paper such an agent framework has been extended to work as a web browser, a web server, or even a web-service provider, in order to communicate with each other in a multi-agent communication fashion.

The meta-logical system for one single agent consists of three main parts: meta-programs for multiple ontologies, a meta-interpreter, and the communication facility. Each of the meta-programs contains meta-level descriptions of ontologies from SW. That is, the ontologies expressed in their native form, e.g. in RDF, RDFS, and OWL languages, are transformed to a meta-logical representation. Some elements in one ontology may be related to some elements in another ontology. The meta-interpreter is the system's inference engine which is used to infer implicit information from the multiple ontologies. The communication facility supports the communication among the individual agents. One block in Fig. 1. illustrates one agent.

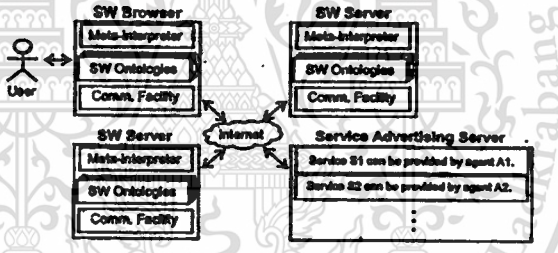


Fig. 1. Our multi-agent communication system

When several agents of this kind are formed as an agent community, the way the multi-agent system works can be explained as follows. Initially the user queries an SW browser to get answers from an SW ontology on SW. The browser can perform two alternative ways. Firstly, it may retrieve this ontology from SW, transform it into a meta-program, and finally reason with the meta-program to infer the answers; if some elements in this ontology are related to some elements of another ontology, the interpreter will try to reason with that ontology in itself (by retrieving it first), or request reasoning of that ontology in an SW server and obtain the answers from that server, and this scenario may repeat itself. Alternatively, the browser passes the query to an SW sever to answer and gets the answers back for the user. The server infers those answers based on its inferential results which sometimes also require support of the inferential results derived from other servers. In case the browser does not know

which SW server can answer that query, it will consult the Service Advertising Server which gathers information telling which SW server can provide what service. The browser then uses this information to communicate with the selected server directly.

3 The Meta-Languages and the Meta-Programs

In this section we develop two meta-languages of an SW ontology by mapping the language elements of RDF, RDFS, and OWL into meta-language terms.

3.1 Language Elements of the Semantic Web Ontology

The language elements of an SW ontology are classes, properties, class instances, and relationships between and among them described in the object level and the meta-level as illustrated in Fig. 2. At the object level, an instance can be an individual or a literal of a domain; and a property is a relationship between individuals, or is an individual's attribute. At the meta-level, a meta-instance can be an individual, property, class, and object-level statement; and a meta-property is a property to describe a relationship between and among meta-instances, or is a meta-instance's attribute. Next we define meta-logical terms to express these elements of the object and meta levels.

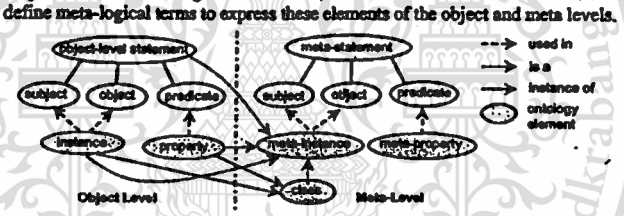


Fig. 2. The elements of an SW ontology at the object level and meta-level

3.2 Meta-languages of the Semantic Web Ontology

In our meta-logical framework, for an SW ontology we distinguish between its object and meta levels, and similarly its object and meta languages. Consequently, we have formulated two meta-languages; one discusses mainly about objects and their relationships, we call it "meta-language for the object level (ML)", and the other, called "meta-language for the meta-level (MML)", discusses mainly about classes, class instances, properties, and their relationships. Due to some connection between the object and meta levels, ML and MML are slightly overlapped.

- **A meta-language for the object level (ML)**

Objects and their relationships at the object level as well as some provability and references at the meta-level are specified in an SW ontology and this information is

expressed at the meta-level by the elements below. (Note that the linguistic elements of provability are a part of AgentML (see section 5) and the linguistic elements expressing references are a part of MML.)

Meta-constant specifies a name of an object and a literal, e.g. `'john'`, including a reference, e.g. a namespace, an ontology name.

Meta-variable stands for a different meta-constant at a different time, e.g. `Person`.

Meta-function symbol stands for the name of a relation between objects, or of an object's property—i.e. an object-level predicate name, such as `'fatherOf'`, `'name'`— and also stand for other meta-level function symbol, e.g. `'←'`, `'∧'`, `':'`, including the name of provability predicate, i.e. `demo`.

Meta-term is either a meta-constant or a meta-variable or a meta-function symbol (probably labeled with a name prefix) applied to a tuple of terms, e.g. `f:'M1'`.

To express an object-level predicate, it has the form: `P(S,O)` where `P` is an object-level predicate name, S and O are meta-constants or meta-variables (we presume all meta-variables appearing in the tuple are universally quantified), e.g. `f:'fatherOf'(f:'M2',f:'M1')`. To express a provability predicate, it has the form: `demo(A,T,P(S,O))`, e.g. `demo(a,o,f:'fatherOf'(f:'M2',f:'M1'))`.

The meta-term expressing an object-level sentence (sometimes with some provability) is a term `P(S,O)` or `demo(A,T,P(S,O))` or a logical-connective function symbol applied to the tuple of these terms. One form of it is a Horn-clause, e.g.

```
f:'fatherOf'(F,Ch) ← demo(b,T,p:'parentOf'(F,Ch)) ∧
                     demo(c,T,m:'male'(F)).
```

Meta-statement for the object level reflects an object-level sentence to its existence at the meta-level. It has the form: `statement(T,object-level-sentence)`, where `T` is an ontology name, e.g.

```
statement(T,f:'fatherOf'(F,Ch) ← demo(b,T,p:'parentOf'(F,Ch)) ∧
                                 demo(c,T,m:'male'(F))).
```

- **A meta-language for the meta-level (MML)**

Additionally, an SW ontology defines classes, properties, and their relationships, and also describes class-instance relations. This information is precisely *meta-information of the object level*. Here we express this information by MML which includes:

Meta-constant specifying a name of an agent, a resource location, a namespace, an ontology, an instance, a property, a class, and a literal.

Meta-variable standing for a different meta-constant at a different time.

Meta-function symbol naming a function, e.g. `port`, `protocol`, `#`.

Meta-term is either a meta-constant or a meta-variable or a meta-function symbol applied to a tuple of terms, e.g. `port(80)`, `protocol(http)`.

In our framework, a name of ontology, class, property, etc., can be referenced by a meta-term in these four forms: `uniquename` or `name:name` or `namespace#name`, e.g. `'http://foo.org'#'family.owl'`, or `location#name`, e.g. `location(protocol(http),port(80),host('foo.org'),path('/'))#'family.owl'`.

Meta-predicate name naming a relation between entities, or a characteristic of a property, which may fall into one of the following categories: class-class relations, class-instance relations, property-property relations, relations between literals and instances/classes/properties, and characteristics of properties [2].

Meta-predicate expressing a relation between entities having the form of `Pred(Sub,Obj)`, or a characteristic of a property having the form of `Pred(Prop)`, where `Pred` is a meta-predicate name, `Sub`, `Obj`, and `Prop` (a property) are meta-terms, e.g. `inverseOf(f:'fatherOf',f:'childOf')`. Let all the meta-variables appearing in a meta-predicate be universally quantified.

Meta-operator expressing a set operation between classes such as union.

Meta-statement being a meta-predicate or meta-predicates connected by logical connectives. One form of the meta-statement is a Horn-clause **meta-rule**. Here are some examples of the meta-rules:

```
meta_statement(o,inverseOf(f:'fatherOf',f:'childOf') ← true).
axiom(t,equiClass(C,EC) ← equiClass(C,EC1) ∧ equiClass(EC1,EC)).
```

The second example represents a rule for a mathematical 'axiom'.

3.3 Meta-programs of the Semantic Web Ontology

Each ontology is transformed into a meta-program containing a (sub-)meta-program expressed in ML, called "MP", and/or a (sub-)meta-program expressed in MML, called "MMP". Another meta-program expresses some mathematical axioms, called "AMP". The inference engine often requires AMP to reason with MP and MMP.

- **The meta-program for the object level (MP)**

MP contains information of instances and their relationships in terms of meta-statements for the object level: `statement(T,P(S,O) ← true)` and `statement(T,P(S,O) ← Body)`, where `Body` expresses a conjunction of object-level predicates and some provabilities; the latter is its Horn-clause form. Note that to state that such a meta-statement belongs to an ontology (or theory) `T` we put `T` as the first argument in `statement()`; and this form of ontology labeling will be used henceforth.

- **The meta-program for the meta-level (MMP)**

MMP contains description of classes, properties, their relations, and class-instance relationships in terms of meta-rules. Here is an example of a statement in MMP:

```
meta_statement(T,equiClass(c1,c2) ← true).    // c1 & c2 are equivalent.
```

- **The meta-program for the axioms (AMP)**

AMP contains axioms for classes and properties. They are expressed in the meta-rule form, for example, the axiom about the equivalence of classes is defined as follows.

```
// the equivalent relation of classes is transitive and symmetric.
axiom(T,equiClass(C,EC) ← equiClass(C,EC1) ∧ equiClass(EC1,EC)).
axiom(T,equiClass(C,EC) ← equiClass(EC,C)).
```

4 Single SW Agent Architecture

An agent in our framework is denoted by <Meta-interpreter, Knowledge Base, Communication, Historical Memory, Transformation> whose components are depicted in Fig. 3 and can be described as follows.

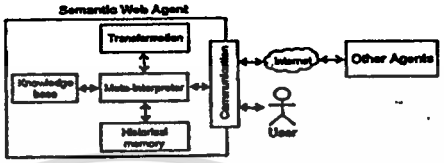


Fig. 3. Single SW Agent Architecture

The Transformation module transforms ontologies obtained from SW into MPs, MMPs, and AMPs. The Knowledge base stores these three kinds of meta-programs. The Meta-interpreter reasons with the meta-programs in order to answer the query posed by the user, and communicates with other agents to get ontologies or answers for a query. Historical memory stores information required for advance reasoning by the meta-interpreter, such as backtracking between alternative answers derived from several agents. Communication module facilitates communication with other agents.

5 Definition of a Communicative Demo

The meta-interpreter is an extension of the one proposed in [2]. The definition of `demo/2` in [2] has been extended to `demo/3`. For `demo(Agent,T,A)`, it means an answer `A` can be inferred from a theory `T` by an agent `Agent`. Here the Vanilla meta-interpreter [1] is adapted for reasoning with multiple ontologies on SW where we identified three kinds of meta-level statements, (1) `statement(T,A←B)`, (2) `meta_statement(T,A←B)` for the meta-level of an ontology, and (3) the mathematical property axioms `axiom(T,A←B)`. The definition of `demo/3` is as follows.

```
demo(_,empty,true).                                              (true)
demo(Agent,T,demo(Agent',T,A)) ← demo(Agent',T,A).               (ref)
demo(Agent,T1∪T2,A) ←                                            (ost)
  statement(T1,A '←' B) ∧ demo(Agent,T2,B).
demo(Agent,T1∪T2,A) ←                                            (mst)
  meta_statement(T1,A '←' B) ∧ demo(Agent,T2,B).
demo(Agent,T1∪T2,A) ←                                            (ast)
  axiom(T1,A '←' B) ∧ demo(Agent,T2,B).
demo(Agent,T1∪T2,A '∧' B) ←                                      (conj)
  demo(Agent,T1,A) ∧ demo(Agent,T2,B).
```

The clauses `(true)`, `(ost)`, and `(conj)` form the Vanilla meta-interpreter. The clause `(ref)` states that when the meta-interpreter tries to prove `demo(Agent,T, demo(Agent',T,A))`, it will prove `demo(Agent',T,A)` by a reflection.

For distributed ontologies, some of them may be referred to in other ontologies. In this case while `demo()` is reasoning with an ontology to derive an answer, this may require it to reason with another unavailable ontology. Therefore, we should add another `demo()` clause to allow `demo()` to retrieve that ontology which is sharable from its location on the web, then transform it into MP and MMP, and can finally reason with it to complete all the inference steps so that it can derive the answer.

```
demo(Agent,T,demo(Agent',T,A)) ←                                    (retr)
  unavailable(T) ∧ myName(Agent') ∧ retrieve(O,location(T)) ∧
  transform(O,P) ∧ demo(Agent',P,A).
```

With this new clause `(retr)`, `demo()` can now work analogously to a web browser.

Additionally, suppose that each server storing an ontology is equipped with this `demo()` definition. Now for the `demo` (at the client) to derive an answer from an unavailable ontology; this can be done easily by that the `demo()` at the client sends the query (for an unavailable ontology) to the server, which has that ontology, to answer the query. For this to be done, we may add two more `demo()` clauses:

```
demo(Agent,T,demo(Agent',T,A)) ←              (certain-agent-comm)
  not myName(Agent') ∧ known(Agent') ∧ unavailable(T) ∧
  agentLocation(Agent',T,Channel) ∧ connect(Agent',Channel) ∧
  communicate(Agent',demo(Agent',T,A)) ∧ disconnect(Agent').

demo(Agent,T,demo(Agent',T,A)) ←           (applicable-agent-comm)
  unknown(Agent') ∧ unavailable(T) ∧
  findAgent(Agent',T,Channel) ∧ connect(Agent',Channel) ∧
  communicate(Agent',demo(Agent',T,A)) ∧ disconnect(Agent').
```

In the last clause, `demo()` searches for an agent who can provide the service by asking this information from a service advertising server (see section 6).

Given all above `demo` clauses, an answer `A` can be inferred from `demo()` in different ways: firstly `A` may be inferred using one or more statement in one or many MP, and/or using meta-statements in MMPs, and/or using axioms in AMP. Alternatively, the inference may require `demo()` to retrieve some ontologies from different sources on SW or to send `demo(Agent,T,A)` to other servers to request for the answer.

- **Meta-language of the agent (AgentML)**

AgentML is a meta-language we use to formulate the agent <Meta-Interpreter, Knowledge Base, Communication, Historical Memory, Transformation>. It discusses about the agent's components, such as the `demo()` definition, the agent name and resources, assumptions in ontologies (this part is connected with MP and MMP), communication methods and facilities, other agents and their ontologies, and so on.

- **Agent creation and the agent's life cycle**

To create and start a new agent, we perform the following steps: (1) assign a unique name to the new agent by asserting `myName(agentName)`; (2) set up its communication channels; (3) start the agent to do an endless observation-action cycle. This means that the agent will keep listening to the communication channels to get a request from the user or from other agents, and response to that request accordingly; when the response is done it returns back to the observation stage again.

6 Multi-agent Communication

An individual agent created by our agent framework can behave in two fashions. One is to work as an SW browser and the other is to work as an SW server. The only difference between them is that the former communicates with a human user and SW servers, whilst the latter communicates to SW browsers and other SW servers. To

follow the way people have used web browsers so far, an SW browser is designed not to receive a query from an SW server or another SW browser although it can do that.

According to the present use of the current web, we expect that a multi-agent community of SW would consist of several SW browsers, SW servers, and a few Service Advertising Servers virtually linked together on the web.

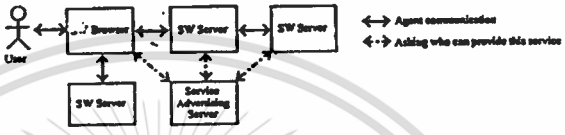


Fig. 4. Multi-agent communication

- **Service Advertising Server**

This server maintains information telling which agent can provide which service in the form of `agentCapability(Agent,channel(Host,Port),Service)`, where `Agent` is a name of a registered agent, `channel(Host,Port)` specifies the channel to communicate with the agent, and `Service` is a service provided by the agent in the form of `service(OntologyName,Reference:PredicateName(_))`.

- **Backtracking among alternative answers**

Backtracking among alternative answers is one basic ability of query answering supported in our multi-agent framework. Suppose an agent A possesses this statement

```
statement(t,f:'fatherOf'(F,Ch) ←
    demo(b,_,p:'parentOf'(F,Ch)) ∧ demo(c,_,m:'male'(F))).
```

in its MP. The user may pose A a query `?demo(a,_,f:'fatherOf'(F,Ch))` to find out who is the father of who. Then to get the answers, A will pose the query `?demo(b,_,p:'parentOf'(F,Ch))` to agent B and `?demo(c,_,m:'male'(F))` to agent C. The answers inferred from B and C will be given to A one at a time, and A will perform backtracking to all the possible answers. This is achieved by each agent employing a historical memory to keep trace of the possible answers.

7 Query Answering by Multi-agent Communication

To illustrate our framework, we use a book purchase scenario of multi-agent communication that works with multiple distributed ontologies and distributed reasoning.

Suppose we have an online bookshop selling books supplied by some publishers. The bookshop and the publishers have their own SW servers which provide information about the books able to be supplied by them. This book information is described by some SW ontologies and there are differences between the ontologies in the SW servers of different book shops and different publishers.

An online book purchase begins with firstly a customer wants to buy some books from the bookshop. He then uses an SW browser to get some book information—i.e. title, short description about the book—(expressed in some ontologies) from the

bookshop SW server. This information helps him decide which titles to buy. Sometimes, he may want to get more information of the interested titles, such as their publishers, book cover types (e.g. paperback, hardcover), and prices before placing an online order with the bookshop server. Suppose this information is not stored in the bookshop server, but the server can request it from some (probably unknown) publisher servers. In Fig. 5, we list only some small parts of the meta-programs, MP and MMP, possessed by a publisher server, the bookshop server, and service advertising information in a service advertising server, respectively.

Publish SW Server

PMP: *Meta-program for the publication ontology*

```
statement(pmp,pmp#p:'bPrice'(pmp#p:'0262631857','$40')←true).
statement(pmp,pmp#p:'bCover'(pmp#p:'0262631857','hard')←true).
statement(pmp,pmp#p:'bPrice'(pmp#p:'0262635828','$27')←true).
statement(pmp,pmp#p:'bCover'(pmp#p:'0262635828','paper')←true).
```

Bookshop SW Server

BMP: *Meta-program for the book ontology*

```
meta_statement(bmp,instanceOf('Genetic Algorithm',
  bmp#b:'GeneticProgramming')←true).
statement(dmp u T, dmp#d:'bookInfo'(Title,Cover,Price)←
  demo(bookShopAgent,T,dmp#d:'bookInfo'(Title,Cover,Price))).
```

DMP: *Meta-program for the documentation ontology*

```
statement(dmp,dmp#d:'bTitle'(pmp#p:'0262631857',
  'Genetic Algorithm')←true).
statement(dmp,dmp#d:'bTitle'(pmp#p:'0262635828',
  'Genetic Algorithm')←true).
statement(dmp u pmp, dmp#d:'bookInfo'(Title,Cover,Price)←
  dmp#d:'bTitle'(ISBN,Title)∧
  demo(_,pmp,pmp#p:'bCover'(ISBN,Cover))∧
  demo(publisherAgent,pmp,pmp#pf'bPrice'(ISBN,Price))).
```

Service Advertising Server

```
agentCapability(
  publisherAgent, channel('www.BookPublisher.com',2000),
  service(pmp,pmp#p:'bCover'(ISBN,Cover))).
```

Fig. 5. The MMP and MP programs for the demonstration

```
?-demo(browser,_instanceOf(X,bmp#b:'GeneticProgramming')).
  X = 'Genetic Algorithm'
?-demo(browser,_dmp#d:'bookInfo'('Genetic Algorithm',
  Cover,Price)).
  Cover = 'hard', Price = '$40';
  Cover = 'paper', Price = '$27'
```

Fig. 6. Query answering with the multi-agent communication

A demonstration of the query answering of the SW browser is shown in Fig. 6. To answer the first query, the SW browser reasons with its ontologies obtained from the bookshop server. However, for the second, the browser adopts BMP's the second statement, and this requires it to pass this query to the bookshop server to answer.

The bookshop server uses DMP's third statement to infer the ISBN from the title; and it then queries an unknown publisher and the publisher `publisherAgent` for the cover type and price respectively. According to this example, since the `'Genetic Algorithm'` book has two editions, i.e. hardcover and paperback, the user may ask to get more answers after the first one is derived and the browser then performs backtracking to get further answers with the similar reasoning steps of the previous one.

8 Related Works

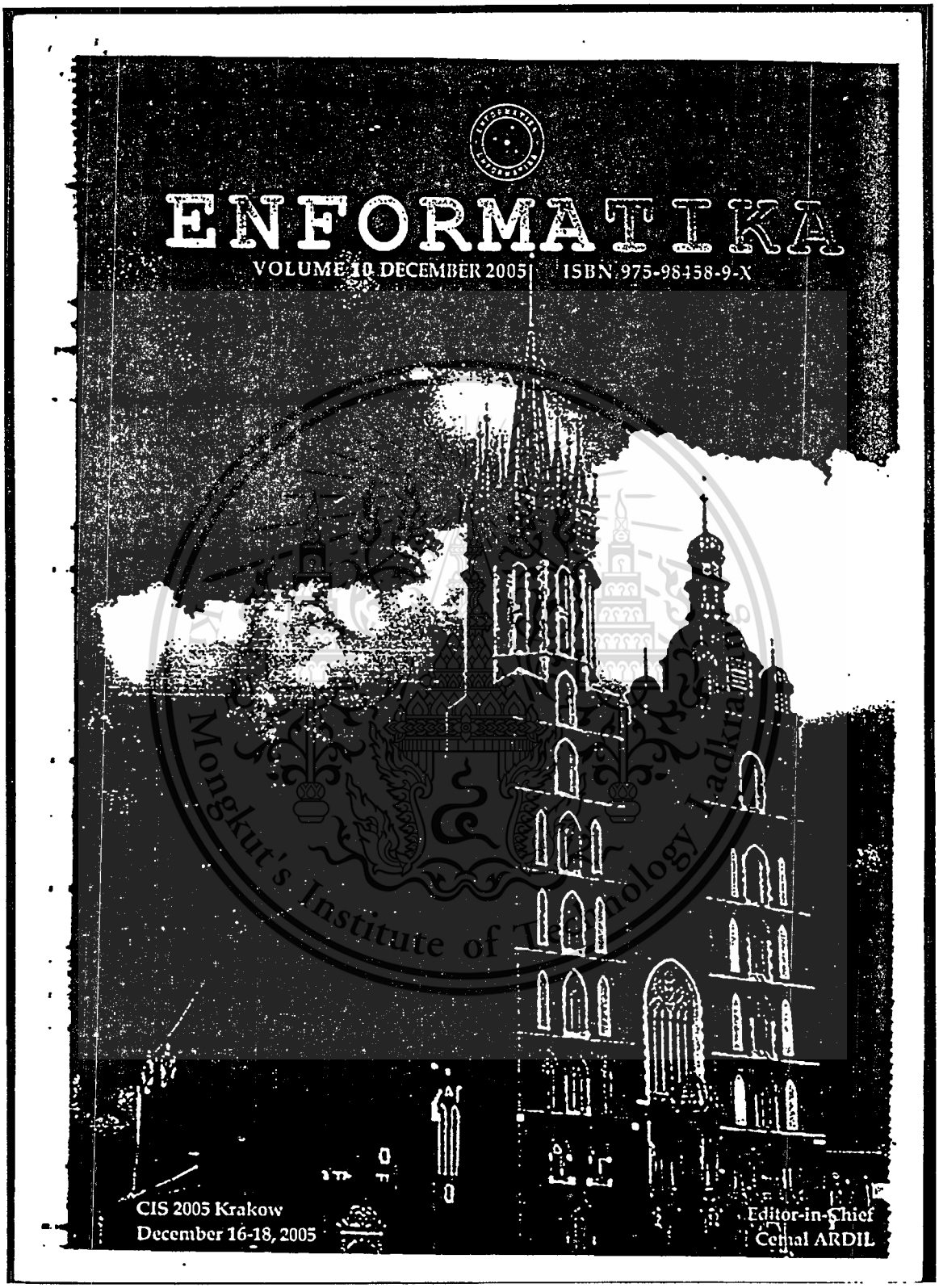
There are some works investigating a multi-agent system adopting SW ontologies. In [3], Serafini et. Tamilin proposed a distributed reasoning architecture for SW which used Distributed Description Logic (DDL) to formulate multiple ontologies interconnected by semantic mappings and used a tableau method for performing inference in DDL. To compare it with our work, here we use meta-logic to represent SW ontologies, and a `demo(.)` predicate to perform inference. We also formalize the `demo(.)` predicate to perform multi-agent communication.

9 Conclusion

We have developed a meta-logical framework for multi-agent communication of SW information. The multiple ontologies represented by meta-programs are reasoned by individual agents using a `demo(.)` predicate. With this framework the ontologies and the information derived from them can be exchanged uniformly among the agents in their community.

References

1. Kowalski, R., A., and Kim, J., S. A Metalogic Programming Approach to Multi-agent Knowledge and Belief, *in AI and Mathematical Theory of Computation*, p. 231-246, 1991.
2. Hirankitti, V., and Tran, X., V. Meta-reasoning with Multiple Distributed Ontologies on the Semantic Web, submitted to the 6th Int. Conf. on Intelligent Technologies, December 2005.
3. Serafini, L., and Tamilin, A. DRAGO: Distributed Reasoning Architecture for the Semantic Web. *In Proc. of the 2nd European Semantic Web Conf.*. LNCS, Vol. 3532, Springer-Verlag, p. 361-376, 2005.
4. Zou, Y., Finin, T., Ding, L., Chen, H., and Pan, R. Using Semantic Web technology in Multi-Agent systems: a case study in the TAGA Trading agent environment. *In Proc. of the 5th Int. Conf. on Electronic Commerce*. ACM Press, p. 95-101, 2003.
5. Grimnes, G., A., Chalmers, S., Edwards, P., and Preece, A. GraniteNights - A Multi-agent Visit Scheduler Utilising Semantic Web Technology. *In Proc. of the 7th Cooperative Information Agents*. LNCS, Vol. 2782, Springer-Verlag, p. 137-151, 2003.
6. Payne, T., R., Singh, R., and Sycara, K. Processing Schedules using Distributed Ontologies on the Semantic Web. *In Proc. of the Int. Workshop on Web Services, E-Business, and the Semantic Web*. LNCS, Vol. 2512, Springer-Verlag, p. 203-212, 2002.



Semantic Web Agent Communication Capable of Reasoning with Ontology and Agent Locations

Visit Hirankitti and Vuong Tran Xuan

Abstract—Multi-agent communication of Semantic Web information cannot be realized without the need to reason with ontology and agent locations. This is because for an agent to be able to reason with an external semantic web ontology, it must know where and how to access to that ontology. Similarly, for an agent to be able to communicate with another agent, it must know where and how to send a message to that agent. In this paper we propose a framework of an agent which can reason with ontology and agent locations in order to perform reasoning with multiple distributed ontologies and perform communication with other agents on the semantic web. The agent framework and its communication mechanism are formulated entirely in meta-logic.

Keywords— Semantic Web, agent communication, ontologies.

I. Introduction

COMMUNICATION of Semantic Web (or briefly "SW") information between browsers and servers can be understood as multi-agent communication. However, this communication cannot actually be realized without the need to reason with ontology and agent locations. This is because for an agent to reason with an external SW ontology, it must know where and how to access to that ontology. Similarly, for an agent to communicate with another agent, it must know where and how to send a message to that agent. To achieve this, in this paper we propose a meta-logical model of SW communication among agents using meta-information of ontology and agent locations.

Some previous works on an agent system related to SW are: Zou et. al. [4] used SW languages, as the languages for expressing agent's messages and knowledge base, to specify and publish common ontologies; [5] presented a multi-agent based scheduling application in which data sources are described by SW languages and encapsulated in the agents. In [6], an agent is built to perform scheduling with distributed ontologies about events, e.g. conferences, classes, published on the SW. Those approaches are mainly related to applying the SW technology in a multi-agent system. However, here we are concerned with multi-agent communication and reasoning with distributed ontologies and some works [1, 2] were done previously. In this paper we have extended the communication framework in [2] to reason with ontology and agent locations.

The rest of this paper is organized as follows. Next we give an overview of our framework. Section III presents our meta-representation of SW ontologies and section IV describes our single agent architecture. Section V describes the meta-interpreter which can reason with ontology and agent locations, and section VI introduces multi-agent communication. Section VII shows how to query and reason with SW ontologies by multi-agent communication. Section VIII covers some implementation issues. Finally, we discuss about other related works and conclude this paper.

II. Our Framework

The meta-logical system for one agent consists of three main parts: meta-programs for multiple ontologies, a meta-interpreter, and the communication facility. Each meta-program contains meta-logical representations of ontologies obtained from the transformation of these ontologies defined in RDF, RDFS, and OWL. Some elements in one ontology may be related to some elements in another. The meta-interpreter is the inference engine for infering implicit information from the multiple ontologies. The communication facility supports the communication among the agents. One block in Fig. 1 illustrates one agent.

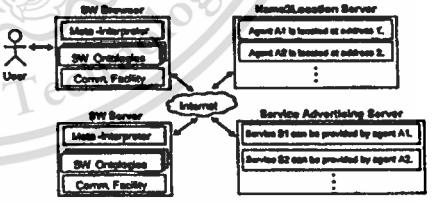


Fig. 1 Our SW multi-agent communication system

When several agents of this kind are formed as a community, the way the multi-agent system works is that initially the user queries an SW browser to get answers from an SW ontology on SW. The browser can perform two alternative ways.

Firstly, it may retrieve this ontology from SW, transform it into a meta-program, and then reason with the program to infer

Manuscript received November 30, 2005. This research was supported by the Japan International Cooperation Agency (JICA) under the ASEAN University Network / Southeast Asia Engineering Education Development Network (AUN/SEED-Net) Program.

Visit Hirankitti is with the Department of Computer Engineering, Faculty of Engineering, King Mongkut's Institute of Technology Ladkrabang, Bangkok, Thailand (corresponding author to provide phone: +66-2-739-2400; fax: +66-2-739-2404; e-mail: visit@ce.kmitl.ac.th).

Vuong Tran Xuan is with the Department of Computer Engineering, Faculty of Engineering, King Mongkut's Institute of Technology Ladkrabang, Bangkok, Thailand (corresponding author to provide phone: +66-2-739-2400; fax: +66-2-739-2404; e-mail: txvuong@yahoo.com).

the answers; if some elements in this ontology are related to some elements of another ontology, the interpreter will try to reason with that ontology in itself (by retrieving it first), or request reasoning of that ontology in an SW server and obtain the answers from that server, and this scenario may repeat itself. For the browser to be able to retrieve an ontology, it must know which server the ontology belongs to, and how and where to access to it. This is the ontology's meta-information provided in the ontology. The browser will use this to contact with that server and request that ontology from it, or to pass a query to that server so that the server can derive an answer from the ontology.

Alternatively, the browser passes the query to an SW server to answer and gets the answers back for the user. The server infers those answers based on its inferential results which sometimes also require support of the inferential results derived from other servers. In case the browser does not know which server can answer that query, it will consult the Service Advertising Server which gathers information telling which server can provide what service. The browser then uses this information to communicate with the selected server directly. For the browser to communicate to any server as said earlier, having known the server name the browser will pass the name to the Name2Location server to obtain the server location and then make contact with that server at that location. Note that conceptually the term 'location' we use here is intended to be an abstract one; an agent location could be the place, such as an address (IP address) on the Internet, or even a (postal) address, where the agent can be reached.

III. The Meta-Languages and Meta-Programs

A. Language Elements of the Semantic Web Ontology

The language elements of ontology are classes, properties, class instances, and relationships between and among them described in the object level and the meta-level as in Fig. 2.

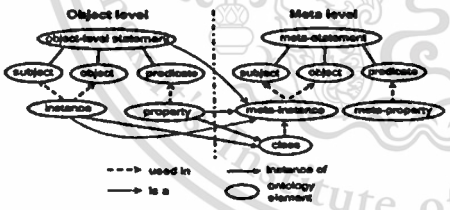


Fig. 2 The elements of an SW ontology at the object level and meta-level

At the object level, an instance can be an individual or a literal of a domain; and a property is a relationship between individuals, or is an individual's attribute. At the meta-level, a meta-instance can be an individual, a property, a class, and an object-level statement; and a meta-property is a property to describe a relationship between and among meta-instances, or is a meta-instance's attribute.

B. Meta-information of Ontologies

To facilitate communication between agents and reasoning with multiple distributed ontologies, language elements of an ontology should be associated with an ontology name. An ontology should also be related to the agent possessing it, the agent's communication channel used to access to that ontology, the file containing this ontology, and the file's path location. This is some *meta-information* of the ontology and it should be treated as a part of a meta-level of the ontology.

C. Meta-languages of the Semantic Web Ontology

In our framework, for an SW ontology we distinguish between its object and meta levels, and similarly its object and meta languages. Hence, we have formulated two meta-languages: one discusses mainly about objects and their relationships we call "meta-language for the object level (ML)" and the other, called "meta-language for the meta-level (MML)", discusses mainly about classes, class instances, properties, and their relationships. MML includes the meta-language representing the meta-information of an ontology discussed earlier in III.B. Due to some connections between the object and meta levels, ML and MML are slightly overlapped.

- A meta-language for the object level (ML)

Objects and their relationships at the object level as well as some provability and references at the meta-level are specified in an SW ontology and this information is expressed at the meta-level by the elements below. (Note that the linguistic elements of provability are a part of AgentML (see section V) and the elements expressing references are a part of MML.).

Meta-constant specifies a name of an object and a literal, including a reference, e.g. a namespace and an ontology name.

Meta-variable stands for a different meta-constant at a different time, e.g. `Person`.

Meta-function symbol stands for the name of a relation between objects, or of an object's property—i.e. an object-level predicate name, such as `'fatherOf'`—including the name of provability predicate, i.e. `demo`. The meta-function symbol also stands for other meta-level function symbol, e.g. `'←'`, `'∧'`, `':'`, `'#'`. Finally the meta-function symbol can also be a term in the form `<ontology_name>:<namespace>#<object-level predicate name>` where `':'` and `'#'` are meta-function symbols, and `<ontology_name>` and `<namespace>` are meta-constants or meta-variables.

Meta-term is either a meta-constant or a meta-variable or a meta-function symbol applied to a tuple of terms, e.g. `'family_ont':'f'#'M1'`.

To express an object-level predicate, it has the form: `P(S, O)` where `P` is an object-level predicate name, `S` and `O` are meta-constants or meta-variables (we presume all meta-variables appearing in the tuple are universally quantified), e.g. `'f'#'fatherOf'('f'#'M2', 'f'#'M1')` To express a provability predicate, it has the form: `demo(A, T P(S, O))`, e.g. `demo(a, o, 'o':'f'#'fatherOf' 'o':'f'#'M2', 'o':'f'#'M1'))`.

The meta-term expressing an object-level sentence (sometimes with some provability) is a term `p(s, o)` or `demo(h, t, p(s, o))` or a logical-connective function symbol applied to the tuple of these terms. One form of it is a Horn-clause, e.g.

```
'o':'f'#'fatherOf'(F, Ch) ←
    demo(b, ob, 'ob':'p'#'parentOf'(F, Ch)) ∧
    demo(c, oc, 'oc':'m'#'male'(F)).
```

Meta-statement for the object level reflects an object-level sentence to its existence at the meta-level. It has the form: `statement(T, object-level-sentence)`, where `T` is an ontology name, e.g.

```
statement(oo∪ob∪oc,'o':'f'#'fatherOf'(F, Ch) ←
  demo(b, ob, 'ob':'p'#'parentOf'(F, Ch)) ∧
  demo(c, oc, 'oc':'m'#'male'(F))).
```

- A meta-language for the meta-level (MML)

Additionally, an SW ontology defines classes, properties, and their relationships, and also class-instance relations. This information is precisely *meta-information of the object level*. Here we express this information by MML which includes:

Meta-constant specifying a name of an agent, a namespace, an ontology, a communication channel, a file's path location, a file, an instance, a property, a class, and a literal.

Meta-variable standing for a different meta-constant at a different time.

Meta-function symbol naming a meta-level function, e.g. `port`, `protocol`, `#`, `:`, `path`, `file`, `location`.

Meta-term is either a meta-constant or a meta-variable or a meta-function symbol applied to a tuple of terms, e.g. `port(80)`, `protocol(http)`, `location(path('/'), file('family.owl'))`.

In our framework, a name of a class, a property, etc., can be referenced by a meta-term in these three forms: `uniqueName` or `namespace#name` or `ontologyName:namespace#name`, e.g. `'owl'#'inverseOf'`, `'o':'f'#'fatherOf'`.

Meta-predicate name naming a relation between entities, or a characteristic of a property, which fall into one of the following categories: class-class relations, class-instance relations, property-property relations, relations between literals and instances/classes/properties, and characteristics of properties [2]. A predicate name is labeled with a term to be associated with its namespace and the name of an ontology it belongs to.

Meta-predicate expressing a relation between entities of the form `Pred(Sub,Obj)`, or a characteristic of a property in the form `Pred(Prop)`, where `Pred` is a meta-predicate name, `Sub`, `Obj`, and `Prop` (a property) are meta-terms, e.g. `'owl'#'inverseOf'('o':'f'#'fatherOf', 'o':'f'#'childOf')`. Let all the meta-variables appearing in a meta-predicate be universally quantified.

Meta-operator expressing a set operation between classes such as union, intersection.

Meta-statement being a meta-predicate or meta-predicates connected by logical connectives. One form of the meta-statement is a Horn-clause meta-rule. Here are some examples of the meta-rules:

```
meta_statement(o, 'owl'#'inverseOf'('o':'f'#
  'fatherOf', 'o':'f'#'childOf') ← true).

axiom(t, 'owl'#'equivalentClass'(C, EC) ←
    'owl'#'equivalentClass'(C, EC1) ∧
    'owl'#'equivalentClass'(EC1, EC)).
```

The second rule represents a mathematical 'axiom'.

- A meta-language for the meta-information of ontologies

A meta-language expressing the meta-information of ontologies discussed earlier in the section III.B is also included in MML, although it could be taken to be at a higher meta-level than MML; but for the simplicity we did not do that. The meta-information relates an ontology to its agent, the communication channel used to access to it, a file's path location, and the file that contains it; this meta-information is formulated in MML in the form of `meta_info_statement(ontology, agent, port, protocol, location(path, file))`, e.g. `meta_info_statement(dmp, bookShopAgent, 80, http, location('/', 'DocOnto.owl'))`.

D. Meta-programs of the Semantic Web Ontology

Each ontology is transformed to a meta-program containing a (sub-)meta-program expressed in ML, called "MP", and/or a (sub-)meta-program expressed in MML, called "MMP". Another meta-program expresses some mathematical axioms using MML, called "AMP". The inference engine often requires AMP to reason with MP and MMP.

- The meta-program for the object level (MP)

MP contains meta-statements for the object level: `statement(T, P(S, O) ← true)` and `statement(T, P(S, O) ← Body)`, where `Body` expresses a conjunction of object-level predicates and some provability; the latter is its Horn-clause form. Note that to state that a meta-statement belongs to an ontology `T` we put `T` as the first argument; and this form of ontology labeling will be used henceforth.

- The meta-program for the meta-level (MMP)

MMP contains description of classes, properties, their relations, and class-instance relationships in terms of meta-rules. It also contains statements expressing ontology meta-information. Here is an example of a statement in MMP:

```
meta_info_statement(dmp, bookShopAgent, 80,
  http, location('/', 'DocOnto.owl')).
```

- The meta-program for the axioms (AMP)

AMP contains axioms for classes and properties. For example, an axiom defining an equivalence of classes:

```
axiom(T, 'owl'#'equivalentClass'(C, EC) ←
    'owl'#'equivalentClass'(C, EC1) ∧
    'owl'#'equivalentClass'(EC1, EC)).
```


IV. Single SW Agent Architecture

An agent in our framework is denoted by <Meta-Interpreter, Knowledge Base, Communication, Historical Memory, Transformation> whose components are depicted in Fig. 3. The Transformation module transforms ontologies obtained from SW to MPs, MMPs, and AMPs, and the knowledge base stores them. The Meta-interpreter reasons with them in order to answer queries posed by the user, and communicates with other agents to get ontologies or answers for queries. The Historical Memory stores information required for advance reasoning by the meta-interpreter. The Communication module facilitates communication with other agents.

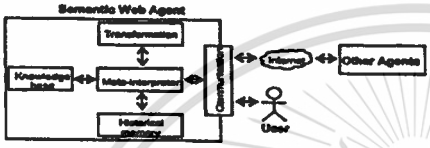


Fig. 3 Single SW Agent Architecture

- Meta-language of the agent (AgentML)

AgentML is a meta-language we use to formulate the agent. It discusses about the agent's components, such as the demo(.) definition, the agent's name and resources, assumptions in ontologies (this part is connected with MP and MMP), communication methods and facilities, its locations used for communication, other agents and their ontologies, and so on.

V. A Communicative Demo

The demo predicate [7] is used as our meta-interpreter. Our demo definition, which can reason with multiple distributed ontologies and communicate with other agents proposed in [1, 2], has been extended here to reason with ontology and agent locations in order to realize its task of communication of SW information. For demo(Agent,T,A), it means an answer A can be inferred from a theory T by an agent Agent. In [2] the Vanilla is adapted for reasoning with multiple ontologies where we identified three kinds of meta-level statements, (1) statement(T,A ← B), (2) meta_statement(T,A ← B) for the meta-level of an ontology, and (3) the mathematical axioms axiom(T,A ← B). The definition of demo/3 is:

```
demo(_,empty,true).                                          (true)

demo(Agent,T1∪T2,A '∧' B) ←                                  (conj)
  demo(Agent,T1,A) ∧ demo(Agent,T2,B).

demo(Agent,T,demo(Agent',T,A)) ←                             (ref)
  demo(Agent',T,A).

demo(Agent,T1∪T2,A) ←                                        (ost)
  statement(T1,A '←' B) ∧demo(Agent,T2,B).

demo(Agent,T1∪T2,A) ←                                        (mst)
  meta_statement(T1,A '←' B) ∧
  demo(Agent,T2,B).

demo(Agent,T1∪T2,A) ←                                        (ast)
  axiom(T1,A '←' B) ∧ demo(Agent,T2,B).
```

The clauses (true), (ost), and (conj) form the Vanilla. The clause (ref) states that when the meta-interpreter tries to prove demo(Agent,T,demo(Agent',T,A)), it will prove demo(Agent',T,A) by a reflection.

For distributed ontologies, some ontologies may be referred to in others. In this case while demo is reasoning with an ontology to derive an answer, this may require it to reason with another unavailable ontology. So we add the following clause to allow demo to retrieve that ontology from its location on the web, transform it into MP and MMP, and then reason with it to complete all the inference steps so that it can derive the answer.

```
demo(Agent,T,demo(Agent',T,A)) ←                             (retr)
  myName(Agent') ∧ unavailable(T) ∧
  O:NS#Goal = A ∧
  meta_info_statement(
    O,Agent'',Port,Channel,Location) ∧
  retrieve(O,Agent'',Port,Channel,Location) ∧
  transform(O,P) ∧ demo(Agent,P,A).
```

With this clause, demo can work analogously to a browser.

Additionally, when each server storing an ontology is equipped with this demo definition, for demo (at the client) to derive an answer from an unavailable ontology, this can be done by that the demo sends the query (for an unavailable ontology) to the server, which has that ontology, to answer the query. For this to be done, we may add two more demo clauses:

```
demo(Agent,T,                                        (certain-agent-comm)
    demo(Agent',T,A)) ←
  not myName(Agent') ∧ known(Agent') ∧
  unavailable(T) ∧
  agentLocation(Agent',Location,
                Port,Channel) ∧
  connect(Location,Port,Channel,ConnectID) ∧
  communicate(ConnectID,demo(Agent',T,A)) ∧
  disconnect(ConnectID).

agentLocation(Agent,Addr,Port,Ch) ←
  connect(www.n21.net,80,http,ConnectID) ∧
  communicate(ConnectID,demo(www.n21.net,T,
    name_location(Agent,Addr,Port,Ch))) ∧
  disconnect(ConnectID).

demo(Agent,T,                          (applicable-agent-comm)
    demo(Agent',T,A)) ←
  unknown(Agent') ∧ unavailable(T) ∧
  findAgent(Agent',A) ∧
  demo(Agent,T,demo(Agent',T,A)).

findAgent(Agent,Goal) ←
  agentLocation(sa_sever,Location,
                Port,Channel) ∧
  connect(Location,Port,Channel,ConnectID) ∧
  communicate(ConnectID,demo(sa_server,_,
    agentCapability(Agent,Service))) ∧
  matchOK(Goal,Service) ∧
```

`disconnect(ConnectID).`

In this clause, the `demo` searches for an agent who can provide the service by asking a service advertising server (see section VI).

Given all the above clauses, `A` can be inferred from `demo` in different ways: firstly `A` may be inferred using statements in one or many MPs, and/or using meta-statements in MMPs, and/or using axioms in AMP. Alternatively, the inference may require `demo` to retrieve some ontologies from different sources on SW or to send `demo(Agent,T,A)` to other servers to request for the answer.

- **Agent creation and the agent's life cycle**

To create and start a new agent, we perform: (1) assign a unique name to the agent by asserting `myName(agentName)`; (2) set up its communication channels; (3) register its name, locations, ports, and channels to the Name2Location server; (4) start the agent to do an endless observation—action cycle—listening to the communication channels to get a request from the user or other agent, and responsing to that request accordingly; when the response is done it returns back to the observation stage again.

VI. Multi-Agent Communication

An individual agent created by our agent framework can behave in two fashions. One is to work as an SW browser and the other is to work as an SW server. The only difference is that the former communicates with a human user and SW servers, whilst the latter communicates with SW browsers and other SW servers. Due to the usage of the current web, we expect that a multi-agent community of SW would consist of SW browsers, SW servers, Name2Location servers and Service Advertising servers virtually linked together on the web (see Fig. 1).

A Name2Location Server provides a communication location of an agent when being asked with an agent name. It has the fixed address: `'www.n2l.net'`. It possesses the facts in the form of `name_location(Agentname, AgentAddress, Port, Channel)`.

A Service Advertising server has the name `'sa_server'`. It maintains information telling which agent can provide which service in the form of `agentCapability(Agent, Service)`, where `Agent` is a name of a registered agent and `Service` is a service provided by the agent in the form of `OntologyName:Namespace#PredicateName(...)`.

VII. The Query Answering

To illustrate our framework, we use a book purchase scenario. Suppose we have an online bookshop selling books supplied by some publishers and providers. The bookshop, the publishers, and the providers have their own SW servers which provide information about the books able to be supplied by them. This information is described by some ontologies and there are differences between the ontologies in the servers of different bookshops, different publishers, and different providers.

An online book purchase begins with a customer wants to buy some books from a bookshop. He then uses an SW browser to get some book information—i.e. title, short description about the book—(expressed in some ontologies) from a bookshop SW server. This information helps him decide which titles to buy. Sometimes, he may want to get more information of the interested titles, such as publishers, book cover types (e.g. paperback, hardcover), and prices before placing an order with the bookshop server. Suppose this information is not stored in the bookshop server, but the server can request it from some (probably unknown) publisher servers and/or provider servers. In Fig. 4, we list only some parts of the meta-programs, MP and MMP, possessed by a publisher server, a provider server, the bookshop server, and also a part of service advertising information in a service advertising server, respectively.

A demonstration of the query answering of the SW browser is shown in Fig. 5. To answer the first query, the SW browser reasons with its ontologies obtained from the bookshop server. However, for the second, the browser adopts BMP's the fourth statement, and this requires it to pass this query to the bookshop server to answer. The bookshop server uses DMP's fifth statement to infer the ISBN from the title; and it then queries an unknown publisher and the provider `providerAgent` for the cover type and price respectively. That is, for the book cover, the bookshop server does not know which agent to ask but for the book price, the book shop server knows that it may ask the `providerAgent` server. For both cases, the bookshop server has to consult to the service advertising server to find the locations and services of these servers and to post its corresponding queries to them and get the answers back. The bookshop server then returns all the answers to the SW browser for presenting to the user.

Publish SW Server

PMP: *Meta-program for the publication ontology*
`meta_info_statement(pmp,publisherAgent,80,http,location('/','PublisherOnto.owl')).`
`statement(pmp,'pmp':'p'#'bCover'('pmp':'p'#'0262635828','hard') ← true).`

Provider SW Server

PPMP: *Meta-program for the publication provider ontology*
`meta_info_statement(ppmp,providerAgent,80,http,location('/','PubProviderOnto.owl')).`
`statement(ppmp,'ppmp':'pp'#'bPrice'('pmp':'p'#'0262635828','$40') ← true).`

Bookshop SW Server

BMP: *Meta-program for the book ontology*

```
meta_info_statement(bmp,browser,80,http,location('/','BookOnto.owl')).
meta_info_statement(dmp,bookShopAgent,80,http,location('/','DocOnto.owl')).
meta_statement(bmp,'rdf'#'type'(
  'Genetic Algorithm','bmp':'b'#'GeneticProgramming') ← true).

statement(dmp u T,'dmp':'d'#'bookInfo'(Title,Cover,Price) ←
  demo(bookShopAgent,T,'dmp':'d'#'bookInfo'(Title,Cover,Price))).
```

DMP: *Meta-program for the documentation ontology*

```
meta_info_statement(dmp,bookshopAgent,80,http,location('/','DocOnto.owl')).
meta_info_statement(pmp,_,80,http,location('/','PublicationOnto.owl')).
meta_info_statement(ppmp,providerAgent,80,http,location('/','PubProviderOnto.owl')).
statement(dmp,'dmp':'d'#'bTitle'('pmp':'p'#'0262635828','Genetic Algorithm') ← true).
statement(dmp u pmp u ppmp,'dmp':'d'#'bookInfo'(Title,Cover,Price) ←
  'dmp':'d'#'bTitle'(ISBN,Title) ∧
  demo(_,pmp,'pmp':'p'#'bCover'(ISBN,Cover)) ∧
  demo(providerAgent,ppmp,'ppmp':'pp'#'bPrice'(ISBN,Price))).
```

Service Advertising Server

```
agentCapability(publisherAgent, 'pmp':'p'#'bCover'(ISBN,Cover)).
agentCapability(providerAgent, 'ppmp':'pp'#'bPrice'(ISBN,Price)).
```

Fig. 4 The MMP and MP programs for the demonstration

```
?- demo(browser,_,'rdf'#'type'(X,'bmp':'b'#'GeneticProgramming')).
   X = 'Genetic Algorithm'
?- demo(browser,_,'dmp':'d'#'bookInfo'('Genetic Algorithm',Cover,Price)).
   Cover = 'hard', Price = '$40'
```

Fig. 5 Query answering with the multi-agent communication

VIII. Implementation Issues

To state an ontology location in our framework, in the following we give an example of how the declaration looks like (see section VII) in OWL as follows:

```
<owl:Ontology rdf:about="dmp">
  <OntologyReferences>
    <Ontology rdf:resource="pmp">
      <path>/</path>
      <file>PublicationOnto.owl</file>
      <port>80</port>
      <protocol>http</protocol>
    </Ontology>
    <Ontology rdf:resource="bmp">
      <agentName
        rdf:resouce="providerAgent"/>
      <path>/</path>
      <file>PubProviderOnto.owl</file>
      <port>80</port>
      <protocol>http</protocol>
    </Ontology>
  </OntologyReferences>
</owl:Ontology>
```

After the transformation, we get an MMP fragment:

```
meta_info_statement(dmp,bookshopAgent,80,
  http,location('/','DocOnto.owl')).
meta_info_statement(pmp,_,80,http,
  location('/','PublicationOnto.owl')).
meta_info_statement(bmp,providerAgent,80,
  http,location('/','PubProviderOnto.owl')).
```

This kind of declaration is used throughout the paper to support the meta-information concerning ontology locations.

IX. Related Works

Some works investigated a multi-agent system adopting SW ontologies. In [3], Serafini et. Tamilin proposed a distributed reasoning architecture for SW using Distributed Description Logic (DDL) to formulate multiple ontologies interconnected by semantic mappings and a tableau method for performing inference in DDL. To compare it with our work, here we use meta-logic to represent SW ontologies, and a `demo(.)` predicate to perform the inference. We also formulate the predicate to be able to reason with ontology and agent locations in order to perform multi-agent communication for SW.

X. Conclusion

We have developed a meta-logical framework for agent communication of SW information. Our agent can reason with distributed ontologies while exchanging the SW information with other agents. The agent can do this by adopting a `demo` predicate which can reason with ontology and agent locations.

References

[1] Hiraishi, V., and Tran, X. V. Meta-reasoning with Multiple Distributed Ontologies on the Semantic Web. To appear in *Proc. of the 6th Int. Conf. on Intelligent Technologies*, December 2005.

[2] Hiraishi, V., and Tran, X. V. A Meta-logical Approach for Multi-agent Communication of the Semantic Web Information. In *Proc. of the 16th International Conference on Applications of Declarative Programming and Knowledge Management*, Japan, October 2005, pp. 7-16.

[3] Serafini, L., and Tamilin, A. DRAGO: Distributed Reasoning Architecture for the Semantic Web. In Proc. of the 2nd European Semantic Web Conf., LNCS, Vol. 3532, Springer-Verlag, pp. 361-377, 2005.

[4] Zou, Y., Finin, T., Ding, L., Chen, H., and Pan, R. Using Semantic Web technology in Multi-Agent systems: a case study in the TAGA Trading agent environment. In Proc. of the 5th Int. Conf. on Electronic Commerce. ACM Press, pp. 95-101, 2003.

[5] Grimnes, G. A., Chalmers, S., Edwards, P., and Preece, A. GraniteNights - A Multi-agent Visit Scheduler Utilising Semantic Web Technology. In Proc. of the 7th Cooperative Information Agents. LNCS, Vol. 2782, Springer-Verlag, pp. 137-151, 2003.

[6] Payne, T. R., Singh, R., and Sycara, K. Processing Schedules using Distributed Ontologies on the Semantic Web. In Proc. of the Int. Workshop on Web Services, E-Business, and the Semantic Web. LNCS, Vol. 2512, Springer-Verlag, pp. 203-212, 2002.

[7] Kowalski, R. A., and Kim, J. S. A Metalogic Programming Approach to Multi-agent Knowledge and Belief. In AI and Mathematical Theory of Computation, 1991, pp. 231-246.

