A META-LOGICAL FRAMEWORK FOR REASONING WITH SEMANTIC WEB ONTOLOGIES WITH RULES

MAI XUAN TRANG

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF

MASTER OF ENGINEERING IN COMPUTER ENGINEERING

INTERNATIONAL COLLEGE

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2011

KMITL-2011-IC-M-006-003

หัวข้อวิทยานิพนธ์	เฟรมเวิร์คที่อาศัย Meta-logic สำหรับคิดวินิจฉัย Ontology พร้อมกฎสำหรับเว็บที่สื่อความหมาย
นักศึกษา	Mai Xuan Trang
รหัสประจำตัว	52601101
ปริญญา	วิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชา	วิศวกรรมคอมพิวเตอร์
พ.ศ.	2554
อาจารย์ที่ปรึกษาวิทยานิพนธ์	Asst. Prof. Dr. Visit Hirankitti

บทคัดย่อ

เป้าหมายของเว็บที่สื่อความหมายคือการเพิ่มความสามารถของเว็บปัจจุบันให้เข้าถึงได้ง่ายมากขึ้น รวมถึงถูกประมวลผลได้ เพื่อทำให้ทรัพยากรบนเว็บสามารถถูกตีความหมายได้โดยโปรแกรมคอมพิวเตอร์ที่เรียกว่า "เอเจนต์ชาญฉลาด" การเพิ่มขีดความสามารถนี้ทำได้โดยการใช้วิธีการใหม่ๆ ในการแทนความหมายของข้อมูลข่าวสาร เพื่อให้ความหมายนั้นสามารถถูกวินิจฉัยโดยเอเจนต์ได้ ภาษาสำหรับสร้างเว็บที่สื่อความหมายหลายภาษาได้รับการพัฒนาขึ้น ได้แก่ RDF RDFS และ OWL เพื่อสื่อความหมายข้อมูลข่าวสารในรูปของ Ontology นับถึงปัจจุบันภาษาเหล่านี้ได้ถูกนำมาใช้อย่างเป็นผลสำเร็จ แต่อย่างไรก็ตาม ยังต้องได้รับการปรับปรุงเพิ่มเติมให้สามารถแทนความหมายของกฎได้ เพื่อที่จะสามารถสื่อถึง Ontology ขั้นสูงซึ่งสามารถนำไปใช้ในการอนุมาน (Deduction) ได้

การทำให้ Ontology บนเว็บขยายขีดความสามารถในการสื่อความหมายของกฎได้สร้างความสนใจเป็นอย่างมากในหมู่นักวิจัย ภาษาสำหรับสื่อความหมาย Ontology หลายภาษาได้ถูกนำเสนอ ซึ่งในวิทยานิพนธ์ฉบับนี้จะนำมาศึกษาวิจัยสองภาษาด้วยกัน ภาษาแรกคือ SWRL (Semantic Web Rule Language) ซึ่งเป็นภาษาสำหรับสื่อความหมายกฎที่รวมเอาภาษา OWL ไว้ด้วย และอีกภาษาหนึ่งคือ OWL 2 ซึ่งเป็นที่ภาษาที่มีพัฒนาการมาจากภาษา OWL ให้สามารถรองรับการสื่อความหมายของกฎ โดยเป็นผลงานขององค์กร W3C

ในวิทยานิพนธ์ฉบับนี้เราได้เพิ่มขยายขีดความสามารถของเฟรมเวิร์คเดิมที่อาศัย Metalogic ซึ่งพัฒนาขึ้นมาสำหรับคิดวินิจฉัย Ontology ที่สื่อความหมายโดยภาษา OWL [1, 2, 3, 4] ให้สามารถคิดวินิจฉัย Ontology ที่สื่อความหมายโดยภาษา SWRL และ ภาษา OWL 2 ตามลำดับ โดยที่ภาษาระดับเมต้าของเฟรมเวิร์คนั้นได้ถูกขยายความสามารถให้สามารถสื่อความหมายของ Ontology ในภาษา SWRL และ OWL 2 ได้ นอกจากนี้ยังมีความจำเป็นที่ต้องเพิ่มเติมในส่วนของ Axiom ในระดับเมต้าเพื่อให้เฟรมเวิร์คเดิมสามารถรองรับการคิดวินิจฉัย Ontology ในภาษา SWRL และ OWL 2 ได้ พร้อมกันนี้ในส่วน Inference Engine ที่สร้างมาจาก demo(.) predicate ยังได้รับการปรับปรุงเช่นกัน ในส่วนท้ายของวิทยานิพนธ์ฉบับนี้เราได้แสดงให้เห็นถึงความสามารถของเฟรมเวิร์คใหม่ที่พัฒนาขึ้นในการคิดวินิจฉัยที่ดีกว่าวิธีการอื่นๆ ที่ได้นำมาศึกษาเปรียบเทียบ

Thesis Title	A Meta-logical Framework for Reasoning with Semantic Web Ontologies with Rules
Student	Mai Xuan Trang
Student ID.	52601101
Degree	Master of Engineering
Program	Computer Engineering
Year	2011
Thesis Advisor	Asst. Prof. Dr. Visit Hirankitti

ABSTRACT

The purpose of the semantic web is to augment the current web by making web resources more accessible and computable, so that the web resources can be interpreted by computer programs, namely 'intelligent agents'. This enhancement is achieved by adopting a new way of presenting information so that its meaning can be reasoned by the agents. Many semantic web languages, such as RDF, RDFS, and OWL, have already been developed to express information in the form of meaningful ontologies. So far these languages have been used successfully, however, they need to be extended with rule representation to be able to express more advanced form of ontology upon which deduction can be applied.

The extension of semantic web ontology with rule representation has recently drawn much attention from the research community, many ontology languages have been proposed for this. In this thesis we have investigated two of them. The first is SWRL (Semantic Web Rule Language), a language for rule formulation which includes OWL in it. The other is OWL 2, a new revision of OWL, which has been recently updated by W3C and it has an improvement over its predecessor especially to support rule formulation.

In this thesis we extend a meta-logical framework, which has been developed for reasoning with OWL ontologies [1, 2, 3, 4], in order to reason with SWRL and OWL 2 ontologies with rules respectively. The meta-languages in that framework are also extended to be able to express SWRL and OWL 2 ontologies (with rules). New meta-level axioms are need to be added to the framework to support reasoning on SWRL and OWL 2 ontologies, and a slightly improved inference engine based on demo(.) predicate is also proposed for this extension. Finally we can argue that our approach is more powerful than others in terms of reasoning ability.

ACKNOWLEDGEMENTS

This thesis is the result of two years of work whereby I have been accompanied and supported by many people. It is great pleasure to acknowledge the debt I owe to all of them.

First of all, I would like to express my sincere thanks to my supervisor, Asst. Prof. Dr. Visit Hirankitti, School of Computer Engineering, Faculty of Engineering, King Mongkut's Institute of Technology Ladkrabang. I am very much indebted to him for his support, guidance, patience and kindness. His influence on me towards logic bring me to the new world—the world of semantic web—, at least from a student who did not know anything about semantic web to an admirer of it. I am also very grateful to him for his struggle of getting through this thesis.

I would like to express my gratitude to people in the International College, especially to lecturers for their lectures and encouragement.

My student life at KMITL was made possible by many friends in Intelligent Communication & Transportation Laboratory. I would like to name particularly Parada Naksook, Pana Jantivas, Worawech Chatsri for their friendships and useful discussion.

My love is expressed to my family in Vietnam for their constant support, understanding and encouragement. I have been always full of energy for my study in their warm arms.

The financial support of this study was generously funded by the Japan International Cooperation Agency (JICA) under AUN/Seed-Net project. I would also like to warmly thank to the secretariats of the AUN/Seed-Net project who have always given helpful and kind support.

Last but not least, I would not forget to thank people in Hanoi University of Science and Technology for their kind and essential help, in particular Prof. Ha Duyen Tu, Assoc. Prof. Dr. Huynh Quyet Thang, Dr. Ta Anh Tuan, to name a few.

Bangkok, Thailand — Mai Xuan Trang

April, 2011

Contents

List of Tables

List of Figures

ABBREVIATIONS/CONVENTIONS

AI	Artificial Intelligence
AMP	Meta Program for Axioms
CARIN	A Representation Language Combining Horn rules and Description Logics
DAML	DARPA Agent Markup Language
DL $\mathcal{ALC}$	A centrally important Description Logic from which comparisons with other varieties can be made
DL	Description Logic
DL $\mathcal{SHOIN}$	A more expressive DL than $\mathcal{ALC}$, It is foundation for OWL-DL
DL $\mathcal{SROIQ}$	A more expressive DL than $\mathcal{SHOIN}$, It is foundation for OWL 2
DLP	Description Logic Program
FOL	First Order Logic
HTML	HyperText Markup Language
IRIs	Internationalized Resource Identifiers
LP	Logic Programming
MAC-SWI	Meta-logical Framework for Agent Communication of Semantic Web Information
ML	Meta Language for the Object Level
MML	Meta Language for the Meta-level
MMP	Meta Program for the Meta-level
MP	Meta Program for the Object Level
N3	Notation 3
NAF	Negation As Failure
N-triples	A possible syntax for RDF
OIL	Ontology Inference Language
OWL	Web Ontology Language
OWL 2	Newly Extension of Web Ontology Language
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
RIF	Rule Interchange Format

RuleML	Rule Markup Language
SW	Semantic Web
SweetProlog	A System to Integrate Ontologies and Rules, and translate them into Prolog program which is reasoned by Prolog engine to deduce new knowledge
SWORIER	Semantic Web Ontologies and Rules for Interoperability with Efficient Reasoning
SWRL	Semantic Web Rule Language
URI	Uniform Resource Identifier
URIref	Uniform Resource Identifier Reference
W3C	World Wide Web Consortium
WWW	World Wide Web
XML	Extensible Markup Language

Chapter 1

Introduction

1.1 Problem definition

In a W3C document namely "Realising the Full Potential of the Web" [9], Tim Berners-Lee identified two major objectives that the web should fulfill. The first objective is to enable people to communicate with each other by allowing them to share knowledge. The second one is to incorporate standards and technologies that can help people structure the information they share in a meaningful way. This vision has been known later as the Semantic Web [10].

The Web's provision, which allows people to write online contents for other people, is a de-facto solution that has changed the computer world. Today, the Web has become one of the most important platforms for e-commerce, communication, entertainment, business, education and sharing knowledge by all means. The current web, however, hinders the second objective. Much of the content on the existing Web, the so-called *syntactic Web*, is human but not machine understandable. Furthermore, the exponential growth in the number of web pages as well as a wide range of web accessible data and services make it difficult for program to evaluate the worth of a resource.

The aim of the Semantic Web is to augment the *syntactic Web* by making web resources more accessible and computable, so that the web resources can be interpreted by compute programs namely 'intelligent agents'. This enhancement is achieved by adopting a new way of presenting information so that its meaning can be reasoned by the agents. Many semantic web languages, such as RDF, RDFS, and OWL, have already been developed to express information in the form of meaningful ontologies.

Ontology is an essential part of the Semantic Web as it forms vocabularies and statements for representing knowledge shared across the web. For an ontology to be processed automatically or reasoned logically by computers, it need to be specified formally and declaratively. Several XML-based markup languages have been developed for expressing ontologies, they were influenced by formalisms such as object-oriented approach, first order logic, and Description Logic [5]. A prominent and highly influential representative of such languages is the Web Ontology Language or OWL which became a W3C standard in 2004 [33]

and which has been updated and extended to OWL 2 in 2009 [34]. The formal semantics of OWL is mainly based on Description Logic as an expressive knowledge representation formalism with a particular emphasis on terminological, i.e. scheme-level, modeling. Rule languages, in contrast, provide an alternative paradigm for modeling knowledge with a stronger focus on instances and relations between them. It seems to be a broad consensus in the Semantic Web community that semantic web ontology should be extended with rules in order to enhance content of an ontology and support reasoning capabilities on OWL ontologies. This is reflected in the Semantic Web stack given in the Figure 1.1. The extension of the ontologies with rules leads to the following problems that need to be investigated.

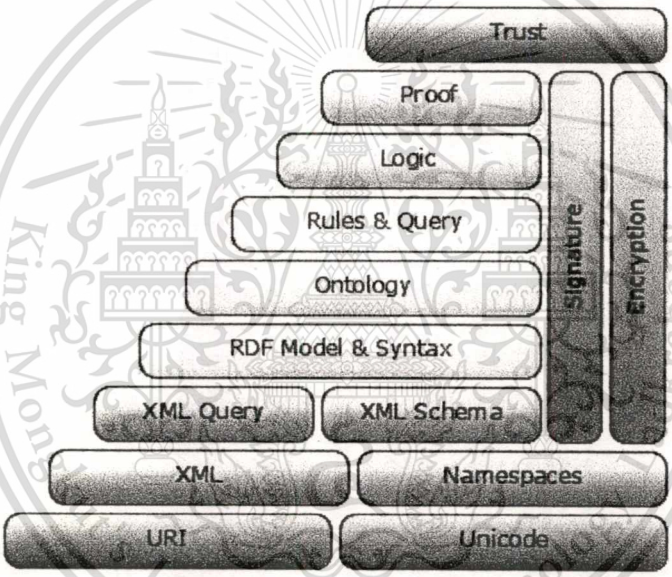


Figure 1.1 Semantic Web stack diagram from W3C

- **Expressing Extension of OWL Ontologies with Rules**

Adding rules to ontologies expressed in OWL could be regarded as an important step in the research on the semantic web, as inferences can now be performed upon semantic web ontologies; and many research proposals have been made. Two alternative ways for introducing rules to semantic web ontologies we consider in this thesis are via SWRL and OWL 2.

While SWRL is an extension of OWL which allows Horn-clause rules to be combined with an OWL ontology, OWL 2 is an improvement of its predecessor—OWL—to have rule formulation based on DL $\mathcal{SROIQ}$. With SWRL and OWL 2, rules can be added to semantic web ontologies. However, we need to relate an ontology with rules in their original form, e.g.

SWRL and OWL2, to its logical forms which form a knowledge base for reasoning mechanisms to derive implicit and explicit information from the ontology.

- **Reasoning with SW Ontologies with Rules**

The use of rule will allow a means to deduce and combine information. Adding rules to semantic web ontologies will increase inferential power of the ontologies and will make it a more viable language for ontology representation. A semantic web ontology extended with rules will allow deduction to perform upon it, so that an inference engine can derive implicit information and extract explicit information from the ontology.

1.2 Motivation

We have realized that a language of semantic web and a language of logic are similar, in that, they are both declarative. The purpose of semantic web is to make semantic web ontologies computable while logic programming aims to make logical sentences computable.

- **Semantic Web ontologies with rules and logic programming**

In logic programming, a domain of discourse is described in terms of objects and their relationship, and on top of that these relationships can be related in terms of rules for deduction. In the semantic web ontologies, just some parts of information of resources are explicitly described by semantic web ontology languages. However, such languages do not provide ways of deriving new knowledge from the ontology. Rules can provide a deductive mechanism to achieve this. Given ontologies together with rules, implicit information from the ontologies can be derived by using an inference engine. Since extension of ontologies with rules analogous to logic programming, a logical approach should be adopted for the reasoning process on ontologies with rules.

1.3 Objectives

The main purpose of this research is to develop a meta-logical framework for reasoning with semantic web ontologies with rules that supports semantic web Ontology languages, and Rule language. The main objectives underlying this goal are:

- To study semantic web languages which are used to express semantic web ontologies with rules. These languages are an extension of OWL DL with the forms of Horn-clause rules represented by SWRL and OWL 2 which is based on DL $\mathcal{SROIQ}$ with more features to express rules.
- To propose an approach for representing SW ontologies with rules based on meta-representation which distinguishes between object-level and meta-level of knowledge represented by ontologies and rules.
- To develop a meta-interpreter as an inference engine for reasoning with semantic web ontologies with rules. This meta-interpreter can reason with meta-programs which are transformed from semantic web ontologies with rules based on the meta-representation.
- To characterize and develop a meta-logical framework for SW agents, this framework treats the meta-programs as its knowledge base and the meta-interpreter as its inference engine.

1.4 Research Methodology

The methodology adopted in this research is mainly studying and analyzing semantic web ontology languages which support rules. Basing on this analysis a transformation module is designed in order to transform ontologies with rules into meta logical statements called meta-programs. A meta-interpreter is then developed in order to reason with the meta-programs.

- **Studying the languages expressing ontologies with rules**

There have been proposals defining new languages for expressing ontologies with rules, such as, SWRL, and OWL 2. In this thesis we study these two languages in order to find ways to represent information contained in semantic web ontologies with rules in a form of logical statements. Such logical representation must have a formal syntax and semantic as well as facilitate reasoning ability in our framework.

- **Defining meta-languages for logical representation of ontologies with rules**

According to our study of SWRL and OWL 2, we shall separate our logical language elements into two levels—object-level and meta-level. With this separation to be able to reason

with each level we define a meta-language for each level. Vocabulary, syntax, and semantics of these meta-languages will be clearly defined in this thesis.

- **Developing a meta-interpreter for reasoning with ontologies with rules**

According to our framework, an ontology with rules is transformed into a meta-logical representation. In order to reason with this representation, a meta-interpreter based on Vanilla meta-meta-interpreter [26] is proposed to use as the inference engine. This meta-interpreter can also be used to develop an intelligent agent to reason with semantic web ontologies with rules.

1.5 Contributions

The main contributions of this thesis are:

- In this thesis we propose a meta-logical framework for reasoning with SW ontologies with rules base on the work of Hirankitti & Tran [1, 2, 3, 4].
- One meta-logical framework is proposed to support SWRL ontologies and another to support OWL 2 ontologies.

1.6 Thesis Outline

The remainder of the thesis is organized as follows:

- **Chapter 2 Ontology Markup Languages**

  The background of semantic web languages as well as semantic web rule language is introduced.

- **Chapter 3 Research Works on Semantic Web Ontologies with Rules**

  This chapter gives an introduction to some approaches to integrate semantic web ontologies and rules. Some approaches which translate ontologies and rules into a logic program are also studied in this chapter.

- **Chapter 4 Previous work and its extension**

  This chapter explains the previous work which is a Meta-logical Framework for Agent Communication of Semantic Web Information (or briefly MAC-SWI)

proposed by Hirankitti & Tran [1, 2, 3, 4]. We then propose and extension of this work so that it can reason with semantic web ontologies and rules.

- **Chapter 5 A Meta-logical framework for Reasoning with SWRL Ontologies**

  In this chapter our proposal, the meta-logical framework for reasoning with semantic web ontologies with rules expressed by SWRL, will be described in detail.

- **Chapter 6 A Meta-logical Framework for Reasoning with Semantic Web Ontologies with Rule in OWL 2**

  In this chapter our proposal, the meta-logical framework for reasoning with semantic web ontologies with rules expressed by OWL 2, will be described in detail.

- **Chapter 7 A Unified Meta-logical Framework for Reasoning with Semantic Web Ontologies with Rules**

  This chapter describes a unified meta-logical framework for reasoning with semantic web ontologies with rules.

- **Chapter 8 Comparison with Related Works**

  We provide an evaluation of our work compare to other related works.

- **Chapter 9 Conclusion and Future Work**

  This chapter presents main results archived in the thesis and discusses potential future works.

1.7 Convention

For a briefly refer to the term "semantic web", we will use just "SW" throughout this thesis.

Chapter 2

Ontology Markup Languages

In this chapter we introduce the concepts and elements of the SW Languages whose aim is to increase machine support for interpretation and integration of resources on World Wide Web (WWW). In addition, we introduce the concept of rules to use for Semantic Web.

2.1 The Semantic Web

2.1.1 Introduction

In the early 90s, the technology of the World Wide Web was invented by Tim Berners-Lee. This invention has significantly changed the way human exchange their information around the world. Over the last decade, the World Wide Web (or briefly WWW) has gained astonished success.

At the beginning of the web, it was first thought of as being a platform for sharing and exchanging document and data, and for communication among corporate users. The content of the current web was mainly targeted for human consumption. The information intended to be processed by computers is generally defined by some standards which provide syntaxes for computers just to understand how to display the information.

Generally, the main problem of the current web is that the information on the web is provided only for human consumption in most cases. The machine cannot understand the meaning of the information. The web is highly loaded enormous amount of content such as texts, pictures, movies which are described in a natural language. Unfortunately, this meaningless information is not useful for the machines because they cannot be processed by the machines.

To solve this problem, new researches have been tried to enrich the current web information with machine comprehensible semantics. One of these is the research on semantic web which aims to provide heterogeneous intelligent access, distributed information, software agents mediating between user needs and the information resources available. This support is essential for bringing the web to its full potential as Tim Berners-Lee said "*extended* web *of machine-readable information and automated services that amplify the Web far beyond current capabilities*" [11].

2.1.2 Annotations and Meaning

The vision of the semantic web is to make resources (not just HTML pages, but a wide range of web accessible data and services) more understandable for machines. Machine-understandable annotations are used to describe the contents and meaning of web resources.

2.1.2.1 RDF

The Resource Description Framework (RDF) was proposed by W3C as a formal language for describing structured information. The goal of RDF is to allow data on the web to be able to exchange among applications while still preserving their original meaning. As opposed to HTML and XML, the main intention is not to display documents correctly, but rather to allow for further processing and re-combination of the information. RDF consequently is often viewed as the basic representation format for developing the semantic web. It provides a powerful tool for forming statements about resources and connecting these statements to infer meaning. With RDF, resources and knowledge about them are stored in a simple and flexible way, so that the resources cannot only be viewed by human, but also be processed by applications.

- **RDF graph**

An RDF statement is in the form of *subject – predicate – object* triple. With this triple form, an RDF statement contains three parts which are analogous to the parts in a simple english sentence (*subject – verb – object*). An RDF triple assertion states a relationship, indicated by the predicated, holds between the things denoted by subject and object of the triple. An example of RDF triple, which is a description of sentence "the resource *me* has name *Trang Mai Xuan*", is depicted in Figure 2.1. A set of RDF triples is interpreted as a graph of these resources, with arcs representing relationships between the resources. Figure 2.2 is an example of an RDF graph in which a resource "me" is described as a person with a name, an email address, and a title (in this example they are "Trang Mai Xuan", "trangmx@gmail.com", and "Mr." respectively).

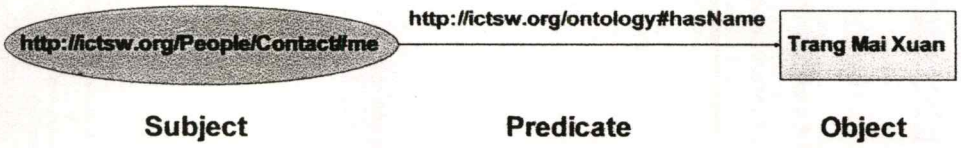


Figure 2.1 An example of RDF triple

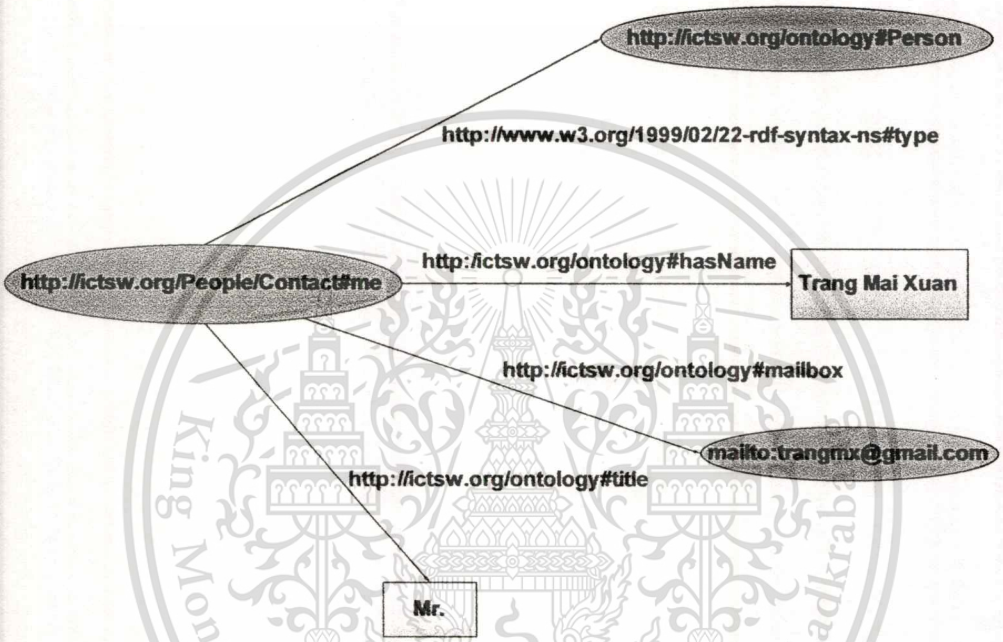


Figure 2.2 An example of RDF graph

In an RDF graph, resources are usually represented by Uniform Resource Identifier references. Uniform Resource Identifiers (URIs) are short strings for identifying Web resources [19]. A URI reference (or briefly URIref) is a URI, together with an optional fragment identifier at the end of the string. For example, the URI reference *http://ictsw.org/People/Contact#me* consists of URI *http://ictsw.org/People/Contact* followed by the # character, and the fragment identifier *me*. A URI resource is uniquely and globally identified. A Resource without URIrefs is called blank node; a blank node indicates an existence of a resource without explicitly referring the URIref of that resource. In each triple the subject is a resource (represented by an ellipse in the graph), the object is either a resource or a literal (represented by rectangle), and the predicate/property is an arc connecting subject and object.

- **RDF syntaxes**

The way of using visual graph for representing RDF is easy for human reading and understanding, but it is clearly not suitable for processing RDF by a computer. In order to express RDF statements in a machine-readable form, there are some RDF syntaxes having been proposed. The first syntax for RDF is called *Notation 3 (N3)* which includes more complex expressions such as paths and rules [9]. The RDF recommendation of 2004, proposed less complicated part of N3 under the name *N-triples* as a possible syntax for RDF. N-triples was further extended to include various convenient abbreviations, leading to the RDF syntax *Turtle*. Figure 2.3 is an example of a Turtle syntax for describing the RDF graph in Figure 2.2.

```
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
@prefix ppl:  <http://ictsw.org/People/Contact#>
@prefix ont:  <http://ictsw.org/ontology#>

ppl:me  rdf:type  ont:Person;
        ont:hasName  "Trang Mai Xuan";
        ont:mailbox  mailto:trangmx@gmail.com;
        ont:title  "Mr.".
```

Figure 2.3 A Turtle syntax example

Here the '@prefix' introduces shorthand identifiers (such as 'ppl:') of XML namespaces. The semicolon ';' after the first line terminates the triple, this also allows us to write many triples for one subject without repeating the name of the subject. In these statements, the annotated resource is ppl:me, which is annotated with four property rdf:type, ont:hasName, ont:mailbox, and ont:title.

The Turtle representation of RDF can be easily processed by a machine but is also readable by human. However, triple representation like Turtle is not the most commonly used RDF syntax in practice due to the fact that many programming languages do not offer standard libraries for processing Turtle syntax. In contrast, essentially every programming language offers libraries for processing XML files, so that application developers can build in existing solutions for XML storage and pre-processing. XML-based serialization RDF/XML, therefore,

becomes the main syntax for RDF. The RDF/XML specification [7] is built upon the XML Information Set [14], XML Namespaces [12], and the XML base recommendation [29] for abbreviating URI references within RDF graphs in their RDF/XML serialization. In order to encode the graph in XML, the nodes and predicates have to be presented in XML terms, that is, in element names, attribute names, element contents, and attribute values. RDF/XML uses QNames as defined in XML Namespaces to represent RDF URI references. Figure 2.4 shows an example for the serialization of the RDF graph in Figure 2.2 as an RDF/XML document.

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

<ont:Person rdf:about=" http://ictsw.org/People/Contact#me">
        <ont:hasName>Trang Mai Xuan</ont:hasName>
        <ont:mailbox rdf:resource="mailto:trangmx@gmail.com"/>
        <ont:title>Mr.</ont:title>
</ont:Person>
</rdf:RDF>
```

Figure 2.4 An RDF/XML document example

As RDF provides only standard syntaxes for describing resources, it does not have any specific mechanism to define a relationship between these resources. So, the question is how can we provide meaning to the Web resources through these annotations? The answer will be via ontologies, which will be introduced in the next section.

2.1.2.2 Semantic Web Ontology

In order to express the meaning of Web resources, ontologies are used for this purpose. Ontology is a term taken form philosophy that refers to the science of describing things in the real world and how they are related. In computer science, ontology is, in general, a 'representation of a shared conceptualisation of a specific domain [20]. In this context, a conceptualisation means an abstract model of some aspect of the world, taking the form of definition of properties of important concepts and relationships. Ontology is designed for the purpose of defining knowledge, reusing, and sharing information effectively. It provides a shared and common vocabulary used in a domain that can be used to communicate between

people, and heterogeneous application systems. In other words, an ontology allows software agents, or in general all computer systems, to share and reuse domain knowledge.

Some important features of an ontology are:

- ability to reuse domain language.
- making domain assumptions explicitly.
- separation of operational knowledge and domain knowledge.
- support of sharing a common understanding of the structure of the information among people and/or software agents, and
- support analysis of domain knowledge.

Ontologies become increasingly important in the fields such as knowledge management, information integration, information retrieval, and electronic ecommerce. SW ontologies are set to play a key role for establishing a common terminology between agents, thus ensuring that different agents have a shared understanding of terms used in semantic markup. For an ontology to be processed automatically or reasoned logically by computers, it needs to be specified formally and declaratively. Several XML-based markup languages have been developed for expressing an ontology such as Resource Description Framework Schema (RDFS), and Semantic Web Ontology Language (OWL). Such languages will be described in more detail in Section 2.1.4. As its name suggests, OWL is the most relevant for ontological modeling. This work is closely related to knowledge representation formalism based on *Description Logics* (DLs) which provides a formal foundation of a major part of OWL standard.

2.1.3 Description Logics

So far Description Logics (DLs) [4] have been important formalisms for ontological modeling. DLs were developed as a family of related knowledge representation ranging from light-weight formalisms to highly expressive logics. They are designed to present and reason about knowledge of an application domain in a structured and formally well-understood way. They are based on a common family of languages, called "description languages", which provide a set of constructors to build concept (class) and role (property) descriptions. Such descriptions can be used in axioms and assertions of DL knowledge bases.

The syntax of a DL is built over the distinct alphabets of *class name* $\mathcal{C}$ (known as *concepts*), *property names* $\mathcal{R}$ (known as *roles*) and individual names $\mathcal{O}$. Depending on the kind

of DL, different constructors are provided to build class expressions (or *classes* for short) and property expression (or briefly *properties*). Intuitively, a class is used to represent a set of individuals, sharing some common characteristics of a domain, and a property is used to present a binary relation between individuals.

Construct Name	Syntax	Symbol	
atomic concept	A		$\mathcal{S}$
universal concept	$\top$	$\mathcal{AL}$	
bottom concept	$\bot$	$\mathcal{AL}$	
atomic role	R		
transitive role	$R \in R_+$		
conjunction	$C \sqcap D$	$\mathcal{AL}$	
disjunction	$C \sqcup D$	$\mathcal{U}$	
negation	$\neg C$	$\mathcal{C}$	
exists restriction	$\exists R.C$	$\mathcal{E}$	
value restriction	$\forall R.C$	$\mathcal{AL}$	
role hierarchy	$R \sqsubseteq S$	$\mathcal{H}$	
inverse role	R^-	$\mathcal{I}$	
unqualified number restriction	$\leq nR$ $\geq nR$	$\mathcal{N}$	
qualified number restriction	$\leq nR.C$ $\geq nR.C$	$\mathcal{Q}$	
nominal	I	$\mathcal{O}$	
complex role inclusions	$S_1 \circ \ldots \circ S_n \sqsubseteq R$	$\mathcal{R}$	

Table 2.1 Overview of DL constructs

An ontology specifies class inclusion relations between its classes and property inclusion relations between its properties. These can be expressed in Description Logics by what so-called *terminological axioms* of the form $X \sqsubseteq Y$ where both X and Y are either class expressions or property expressions. An expression of the form $C \equiv D$ will be called an *equality axiom*. It can be seen as a shorthand for two axioms $C \sqsubseteq D$ and $D \sqsubseteq C$. A set of terminological axioms is often called a 'T-box' in DL knowledge base. Expressions of the form

$a : C$ and $\langle a,b \rangle : R$, where C is a class expression, R is a property expression and a, b are individual names, will be called *DL-atoms*. DL-atoms can be used as axioms called 'assertions', stating class memberships and property memberships. A set of assertions is called 'A-box' in DL knowledge base.

Class expressions, property expressions, and axioms can be seen as an alternative representation of first-order logic (FOL) formulae [18]. More precisely, individuals are equivalent to FOL constants, class expressions correspond to FOL formulae with one free variable, and property expressions are equivalent to FOL formulae with two free variables. In other words, given C is class expression and P is a property expression, C is a class name corresponds to a FOL formula $C(x)$ and P is a property name corresponds to a FOL formula $P(x,y)$ where x, y are free variables. This also applies to expression built by constructors, such as the inclusion axioms are equivalent to the universally quantified implications, i.e. $P \sqsubseteq Q$, where P and Q are property names, corresponding to the FOL formula $\forall x,y R(x,y) \rightarrow S(x,y)$. These mappings are important, because basing on these a logical representation of ontologies and rules is proposed in this thesis.

Description Logics are distinguished by the constructors and the expressive features they support; each DL is named according to a convention introduced in [36], and Table 2.1 overviews DL constructs and their corresponding symbols.

2.1.4 Semantic Web Ontology Languages

After we have introduced description logics, we now introduce their relationship to semantic web ontology languages. The main web ontology language considered in this thesis is OWL [6], which is a W3C recommendation for expressing SW ontologies. The design of OWL was motivated mainly by OIL and DAML+OIL, as well as several other existing languages i.e. RDFS, SHOE and DAML-ONT.

The RDFS (RDF schema) [13], which is recommended by W3C, can be recognized as a simple SW ontology language. RDF and RDFS are the foundation of the SW languages. Next we describe RDFS and then followed by OWL.

2.1.4.1 RDF Schema (RDFS)

The RDF vocabulary description language (RDFS) defines a simple modeling language on top of RDF. RDFS provides the primitives that are required to describe the vocabulary used

in particular RDF models. This description is achieved by expressing set membership of objects in property and class extensions. In RDFS, predefined Web resources rdfs:Class, rdfs:Resource and rdfs:Property can be used to define classes (concepts), resources and properties (roles) respectively.

```
@prefix rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
@prefix rdfs:  < http://www.w3.org/2000/01/rdf-schema#>
@prefix ex:  <http://example.org/Animal#>

ex:Animal      rdf:type     rdfs:Class.
ex:Habitat     rdf:type     rdfs:Class.
ex:Elephant    rdf:type   rdfs:Class; rdfs:subClassOf    ex:Animal.
ex:liveIn  rdf:type  rdfs:Property;
          rdfs:domain   ex:Animal; rdfs:range   ex:Habitat.

ex:south-sahara  rdf:type   ex:Habitat.
ex:Ganesh  rdf:type  ex:Elephant; ex:liveIn  ex:south-sahara.
```

Figure 2.5 An RDFS Ontology

RDFS predefines a set of meta-properties that can be used to present background assumptions in ontologies, they are:

- rdf:type: the instance-of relationship,
- rdfs:subClassOf: the property that models the subsumption hierarchy between classes,
- rdfs:subPropertyOf: the property that models the subsumption hierarchy between properties,
- rdfs:domain: the property that constrains all instances of a particular property to describe instances of a particular class,
- rdfs:range: the property that constrains all instances of a particular property to have values that are instances of a particular class.

RDFS statements are simply RDF triples, that is, RDFS provides no syntactic restriction on RDF triples. Figure 2.5 show a fragment of an animal ontology expressed in RDFS. This fragment defines three classes, i.e., ex:Animal, ex:Habitat, and ex:Elephant (which is rdfs:subClassOf ex:Animal), and a property ex:liveIn, as well as ad domain rdfs:domain and range rdfs:range which are ex:Aninal and ex:Habitat, respectively. In addition, it states that the

resource "ex:Ganesh" is an instance of ex:Elephant, and that it elp:liveIn an ex:Habitat called "ex:south-sahara".

RDFS provides vocabulary for describing light-weight ontologies. However, the expensiveness of RDFS is limited [25]; e.g., we cannot express a statement 'An Elephant is different from a Cow', because negation cannot be expressed in RDFS. This issue led to a specific kind of an Web Ontology Language.

2.1.4.2 OWL (Web Ontology Language)

OWL extends RDFS with new modeling primitives, for example to define properties that are inverses of another, properties that are transitive, properties that are functional, and property restrictions. In essence OWL is based on Description Logics extended by several features to make it suitable for a web ontology language, e.g. using URIs/IRIs as identifiers, import other ontologies, etc. By basing OWL-DL, a sub-language of OWL, on description logics, it can make use of the theory developed for DLs, in particular sophisticated reasoning algorithms.

In OWL, different naming conventions corresponding to description logics are used. OWL classes correspond to concepts in description logics and properties correspond to roles.

OWL has three sub-languages: OWL-Lite, OWL-DL and OWL-Full. OWL-Lite and OWL-DL are very expressive description logics; they are almost equivalent to the $\mathcal{SHIF}$ and $\mathcal{SHOIN}$ DLs [21]. OWL-Full contains features that cannot be expressed in description logics, but it is compatible with RDFS. Precisely OWL-Full can be seen as the union of RDFS and OWL DL.

The latest version OWL 2 is also split in two flavors, OWL 2 DL and OWL 2 Full. OWL 2 DL corresponds to the description logic $\mathcal{SROIQ}$ [22], whereas the full variant is introduced for RDF compatibility. In addition, three profiles were introduced: EL, QL and RL. Each profile imposes usually syntactical restrictions on OWL in order to allow efficient reasoning. OWL 2 EL is aimed at applications which require expressive property modeling and is based on the $\mathcal{EL}$++ logic, which guarantees reasoning with polynomial time w.r.t. ontology size for all standard inference problems. OWL 2 QL is targeted at applications with massive volumes of instance data. In QL, query answering can be implemented on top of conventional relational database systems, and sound and complete conjunctive query answering methods can

be implemented in LOGSPACE. With the EL profile, the standard inference problems run in polynomial time. OWL 2 RL is aimed at scalable applications, which however do not sacrifice to its expressive power. Reasoning algorithms for OWL 2 RL can be implemented by rule-based engine and run in polynomial time. The EL and QL are subsets of OWL 2 DL, whereas RL provides two variants where one is subset of OWL 2 Full and the other one is a subset of OWL 2 DL.

OWL construct	**DL Syntax**	**Manchester syntax**
Thing	$\top$	Thing
Nothing	$\bot$	Nothing
intersectionOf	$C_1 \sqcap \ldots \sqcap C_n$	C_1 and ... and C_n
unionOf	$C_1 \sqcup \ldots \sqcup C_n$	C_1 or ... or C_n
complementOf	$\neg C$	not C
oneOf	$\{x_1\} \sqcup \ldots \sqcup \{x_n\}$	$\{x_1, \ldots, x_n\}$
allValuesFrom	$\forall R.C$	R only C
someValuesFrom	$\exists R.C$	R some C
maxCardinality	$\leq nR$	R max n
minCardinality	$\geq nR$	R min n
cardinality	$\leq nR \sqcap \geq nR$	R exact n
subClassOf	$C_1 \sqsubseteq C_2$	C_1 SubClassOf: C_2
equivalentClass	$C_1 \equiv C_2$	C_1 EquivalentTo: C_2
disjointWith	$C_1 \equiv \neg C_2$	C_1 DisjointWith: C_2
sameAs	$\{x_1\} \equiv \{x_2\}$	x_1 SameAs: x_2
differentFrom	$\{x_1\} \sqsubseteq \neg\{x_2\}$	x_1 DifferentFrom: x_2
domain	$\forall R.\top \sqsubseteq C$	R Domain: C
range	$\top \sqsubseteq \forall R.C$	R Range: C
subPropertyOf	$R \sqsubseteq S$	R SubPropertyOf: S
equivalentProperty	$R \equiv S$	R EquivalentTo: S
inverseOf	$R_1 \equiv R_2^-$	R_1 InverseOf: R_2
TransitiveProperty	$R^+ \sqsubseteq R$	R Characteristics: Transitive
FunctionalProperty	$\top \sqsubseteq \leq 1\, R$	R Characteristics: Functional

Table 2.2 OWL constructs in DL and Manchester OWL syntax

In general, OWL offers convenient constructs that corresponding to description logics, but does not extend their expressiveness. It should be noted that OWL does not make a unique name assumption, so different individuals can be mapped to the same domain element. It allows to express equality and inequality between individual (i.e. a=b, a $\neq$ b) using owl:sameAs and owl:differentFrom respectively. Most algorithms for description logics have already supported this before the OWL specification was created. Not making the unique name assumption is crucial in the semantic web, where it is often the case that many knowledge bases contain information about the same entity. In this case, a common approach is that each knowledge base uses their own URI and owl:sameAs is used to connect them.

Table 2.2 shows some examples of the mapping between constructs in OWL and constructs in description logics. We can see that some features can be mapped directly to description logics, e.g. intersection, and others are syntactic sugar, e.g. functional properties.

OWL has different syntactic formats, in which a knowledge base can be stored. Since it can be converted to RDF, format like RDF/XML or Turtle can be used .There is also a special XML syntax called "OWL/XML" and the Manchester OWL Syntax. The latter one is popular in ontology editors. Examples are shown on the right column in Table 2.2.

2.2 Rules

Rule-based representation is another formalism adopted in logic programming. In SW it has been proposed as a promising approach for modeling knowledge which complements to OWL. In the broadest sense, a rule can be any statement of the form "if the precondition *p* holds then the conclusion *c* holds", where the precondition and the conclusion are logical sentences.

2.2.1 The Syntax

The rules are defined over a first-order vocabulary including disjoint sets of function symbols predicates and variables. The nullary function symbols are called "constants". In contrast to DL, where predicates can only be unary or binary, no restriction is placed on the arity of the predicates. DL does not allow either the function symbols other than constants. The terms are built in the usual way. As usual, atoms have the form $p(t_1,\ldots,t_n)$, where p is an n-ary predicate symbol and $t_1,\ldots,t_n$ are terms.

An ordinary logic program is a set of *rules* each having the form:

$$H \leftarrow B_1 \wedge \ldots \wedge B_m \wedge \sim B_{m+1} \ldots \wedge \sim B_n$$

where H, B_i are classical literals, $n \geq m \geq 0$ and $\sim B_j$ are called negative literals. H is called the head (or consequent) of the rule; $B_1 \wedge \ldots \wedge B_m \wedge \sim B_{m+1} \ldots \wedge \sim B_n$ is called the body (or antecedent) of the rule. $\leftarrow$ is implication connective that is read as "if", so that the whole rule is read as "[head] if [body]", or as "if [body] then [head]", with universal quantification is applied on the whole sentence. A literal $\sim B_i$ intuitively means a *classical negation*, i.e. if B_i is true, then $\sim B_i$ is false. If n=0, then the body is empty, this means *True*, and the connective the "$\leftarrow$" can be omitted. A fact is a rule whose body is empty. A rule is said to be safe if all the variable of the head also appear in some body literals.

Specific classes of programs considered in this thesis included restricted kinds of the above general syntax:

- Definite program, where negated atoms are not allowed in the rules.
- Datalog program is a set of datalog rules which contain conjunctions, constant symbols, and universally quantified variables, but no disjunctions, negations, existential quantifiers, or function symbols.

For exchange the purpose of rules on the web a standard XML encoding for the rules is needed. The effort was undertaken by the RuleML initiative (www.ruleml.org). The proposal RuleML provides a natural mark-up for Datalog rules, using XML tags such as <head>, <body>, <atom>, etc.

2.2.2 Extending Ontologies with Rules

The extension of SW ontologies with rules has recently drawn a lot of attention from the SW research community, and many approaches have been proposed for it. One of them is to combine DL with first-order Horn-clause rules. This is the basis of the Semantic Web Rule Language, SWRL, language for rule formulation and rule extension to OWL. However, inferences on SWRL rules can lead to undecidability even when the rules are assumed to be function-free [24]. In order to make the inferences decidable, some restrictions were put upon the rule language in the form of DL-safe rules [30] or Description Logic Programs DLP [18]. Recently, a new revision of OWL has been developed by W3C, it is called 'OWL 2'. OWL 2 has much improvement over its predecessor—OWL—especially with rule formulation based on DL $\mathcal{SROIQ}$, in which DL rules can be completely internalized as decidable fragment of SWRL. Research works on combining and representing SW ontologies with rules which gave significant influence on our work will be explained in detail in chapter 3.

Chapter 3

Research Works on SW Ontologies with Rules

3.1 Introduction

Many research proposals on adding rules to SW ontologies have been made; they range from hybrid approaches to homogeneous ones. The idea behind the former approaches was that the predicates in the rules and predicates in the ontologies are distinguished, and suitable interface between them is provided. The research works on this direction are such as $\mathcal{AL}$-log language, a hybrid integration of Datalog and Description Logic $\mathcal{ALC}$ [15], CARIN, another hybrid integration of Datalog with different DLs [28], and a hybrid integration of OWL DL (or more precisely the DL $\mathcal{SHOIN}$) with normal rules under answer set semantics [16].

According to the latter approaches both rules and ontologies are combined into the same logical language without making a priori distinction between the rule predicates and the ontology predicates. Two example languages based on these approaches are Description Logic Program (DLP) [18] and Semantic Web Rule Language (SWRL) [24]. In [18] a DLP was proposed by combining Description Logics (DLs), which is the basis for the ontology languages, with Logic Programs (LP), which is the basis for rule languages. It supports a bidirectional translation of premises and inferences between the fragment of DL and LP, and between the fragment of LP and DL. This translation enables one to construct rules on top of SW ontologies. Later SWRL was presented in [24] as a new language for integration of rules and ontologies, in which OWL was extended with Horn-clause rules expressed in RuleML.

Recently SWRL becomes the most commonly used language for describing ontologies with rules. However, the straightforward addition of the rules to ontologies is the main cause of undecidability when reasoning with SWRL ontologies. In order to retain the decidability, some restrictions have been put upon SWRL rules, and this kind of rules can be rewritten as a set of DL axioms using the features introduced in $\mathcal{SROIQ}$ DL. This technique has been presented in [17]. Due to the fact that OWL 2 was developed based on $\mathcal{SROIQ}$ DL, OWL 2 can therefore express rules in the form of DL axioms.

In this chapter we explain two alternative SW languages, SWRL and OWL 2, which are used to express SW ontologies with rules. In this thesis, we propose one reasoning framework to reason with SWRL ontologies with rules, and the other similar framework to reason with OWL

2 ontologies with rules. We also introduce a few systems that transform ontologies with rules into logic programs and then the use interpreters to reason with the logic programs.

3.2 Semantic Web Ontologies with Rules

3.2.1 Semantic Web Rule Language (SWRL)

Whist OWL provides rich vocabulary needed for expressing an ontology, however, adding rules to OWL will make it a more viable language for ontology representation. The basic idea for OWL rules is to extend OWL with a form of rules while maintaining maximum backward compatibility with OWL's existing syntax and semantics; and RuleML (Rule Markup Language) [34] was adopted as the language to express such a rule. Later, a new language SWRL, which is the result of combining OWL DL—the sublanguage of OWL—and RuleML, was introduced.

RuleML is a markup language for publishing and sharing rule bases on the web. Horn clause logic is the foundation for the kernel of RuleML. A rule expressed by RuleML is in the form of $H \rightarrow B_1 \wedge ... \wedge B_k$, where $k \geq 0$, H and B_i are atoms. Each atom is described by using XML tags <atom></atom>. A rule is defined by XML tag <implies> *Body Head* <implies> as in Figure 3.1.

```
<implies>
    <_body>
        <and>
            <atom>atom1</atom>
            ...
            <atom>atomk</atom>
        </and>
    </-body>

    <_head>
        <atom>atom<atom>
    </_head>
</implies>
```

Figure 3.1 RuleML syntax

In the same form with RuleML rule, SWRL rule is in the form of implication consisting of an antecedent and a consequent, where description-logic expressions can occur in both. The intended interpretation of the rule corresponds to that in the classical first-order logic, that is to assert the rule, whenever the conditions specified in the antecedent hold, then the conditions specified in the consequent must also hold; so an SWRL rule is in this form:

Antecedent → *Consequent.*

where *Antecedent*, *Consequent* correspond to *Body*, *Head* in RuleML respectively. Both antecedent and consequent of a rule consist of one or more *atoms*. For example, rules for expressing "hasUncle", "hasMother", "hasFather", "hasSon" relations can be written as follows:

hasParent(x, y) ∧ *hasBrother(y, z)* → *hasUncle(x, z).*

hasParent(x, y) ∧ *woman(y)* → *hasMother(x, y).*

hasParent(x, y) ∧ *man(y)* → *hasFather(x, y).*

hasParent(x, y) ∧ *man(x)* → *hasSon(y, x).*

A rule with multiple atoms in consequent can be transformed into multiple rules. That is, let the multiple atoms in the antecedent form a conjunction B, and multiple atoms in consequent form a conjunction $H_1 \wedge H_2$. We can equivalently express one rule of the form of $B \rightarrow H_1 \wedge H_2$ by the two rules $B \rightarrow H_1$ and $B \rightarrow H_2$ (due to $B \rightarrow H_1 \wedge H_2 \equiv B \rightarrow H_1 \wedge B \rightarrow H_2$).

SWRL abstract syntax was defined by adding axioms to OWL semantics and its abstract syntax in order to allow the rule axioms, and the syntax of the rule axiom is:

axiom ::=rule

rule ::='Implies('{annotation} antecedent consequent)'

antecedent ::= 'Antecedent('{atom}')'

consequent ::= 'Consequent('{atom}')'.

With homogeneous combination of OWL and RuleML, the *atom* can be in the form of either C(x), D(z), P(x,y), Q(x,z), sameAs(x,y), differentFrom(x,y), or builtIn(*pred*,z_1,...,z_n), where C is an OWL DL description, D is an OWL DL data range, P is an OWL DL *individual-valued* property, Q is an OWL DL *data-valued* property, *pred* is built-in relation, x and y are either *individual-valued* variables or OWL individuals, and z, z_1, ..., z_n are either *data-valued* variables or OWL data literals.

Informally, an atom C(x) holds if x is an instance of the class C, an atom D(z) holds if z is a value in the dataRange D, an atom P(x,y) (resp. Q(x,z)) holds if x is related to y (z) by property P (Q), an atom sameAs(x,y) holds if x is interpreted as the same object as y, an atom differentFrom(x,y) holds if x and y are interpreted as different objects, and an atom builtIn(*pred*,z_1,...,z_n) holds if (z_1,...,z_n) is in the extension of the built-in predicate *pred*. Built-in

functions in SWRL [20] is important feature to make the language more flexible when working with datatypes such as numbers or strings.

For example, the rules "hasUncle", "hasMother", "hasFather" can be included to an ontology by adding the following rule axiom to that ontology.

Implies(Antecedent(hasParent(x, y) hasBrother(y, z)) Consequent(hasUncle(x, z))).

Implies(Antecedent(hasParent(x, y) woman(y)) Consequent(hasMother(x, y))).

Implies(Antecedent(hasParent(x, y) man(y)) Consequent(hasFather(x, y))).

SWRL provides both an XML and an RDF concrete syntax. The XML Concrete Syntax is a combination of OWL Web Ontology Language XML Presentation Syntax [17] with RuleML XML syntax. This has several advantages:

- arbitrary OWL classes (e.g., description) can be used as predicate in rules,
- rules and ontology axioms can be freely mixed, and
- interoperability between OWL and RuleML is simplified.

A further advantage of extending OWL's presentation syntax is that the existing XSLT stylesheet can be extended to provide a mapping to RDF graphs that extends the OWL RDF/XML exchange syntax. Full detail of both the XML and the RDF concrete syntax can be found in the SWRL member submission [20]. Figure 3.2 shows an example of the rule uncle written in RDF/XML concrete syntax.

```
<swrl:Imp rdf:ID="Def-Adult">

  <swrl:body rdf:parseType="Collection">
    <swrl:individualPropertyAtom>
      <swrl:propertyPredicate rdf:resource="&eg;hasParent"/>
      <swrl:argument1 rdf:resource="#x1" />
      <swrl:argument2 rdf:resource="#x2" />
    </swrl:individualPropertyAtom>
    <swrl:individualPropertyAtom>
      <swrl:propertyPredicate rdf:resource="&eg;hasBrother"/>
      <swrl:argument1 rdf:resource="#x2" />
      <swrl:argument2 rdf:resource="#x3" />
    </swrl:individualPropertyAtom>
  </swrl:body>

  <swrl:head rdf:parseType="Collection">
    <swrl:individualPropertyAtom>
      <swrl:propertyPredicate rdf:resource="&eg;hasUncle"/>
      <swrl:argument1 rdf:resource="#x1" />
      <swrl:argument2 rdf:resource="#x3" />
    </swrl:individualPropertyAtom>
  </swrl:head>

</swrl:Imp>
```


Figure 3.2 An SWRL rule example

SWRL provides a rule extension to OWL, and this allows rules to be represented with OWL ontologies. SWRL is defined as a syntactic and semantic extension of OWL DL with rules.

3.2.2 OWL 2

- **Limitation of OWL**

OWL has become a popular language for expressing SW ontologies. Although OWL is an expressive language for ontology representation, it is far from being complete. Practicality has revealed OWL's lack of several necessary constructs in both aspects of syntax and expressiveness.

In the syntactic aspect, due to the missing of some constructs to deal with patterns in OWL, in some cases we have to represent knowledge in a verbose manner. For example, OWL allows to state that two subclasses are disjoint, but in order to represent that *several subclasses are pairwise disjoint*, we have to adopt many axioms, each states that the two subclasses are disjoint, this is obviously not very concise.

Another drawback is whilst OWL provides means to assert values of a property for an individual and also to express that two individuals are related by some properties, it does not provide constructs for asserting values that an individual does not have, and also does not provide constructs for expressing that two individuals are not related by some properties.

A major drawback of OWL was also revealed in its limitation to deal with properties. This is due to the fact that whilst OWL includes a relatively rich set of class constructors, its treatment on properties is rather weak, i.e. it lacks of constructs for expressing additional restrictions on properties. In other words, OWL limits the characteristics of properties, i.e. although it is possible to declare properties are transitive, symmetric, functional or inversely functional, it does not allow us to declare a property is asymmetric, reflexive, or irreflexive. A definition of disjoint properties is that some two individuals cannot be connected by two properties which are disjoint, unfortunately this kind of definition is not supported by OWL. In addition, we can define a new class as being a composition of other classes in OWL, but we cannot do so with properties. All these limitations can be overcome by OWL 2, which was developed after OWL by W3C [39].

- **New features in OWL 2**

OWL 2 goes beyond the original OWL by adding to it some new features in order to solve the problems of syntax and expressiveness in OWL. As referred to in the previous section, some patterned knowledge is difficult to express in OWL. To overcome it, OWL 2 adds new constructs to make some common patterns easier to express. For instance, to declare that multiple classes are pairwise disjoint, with OWL several axioms are needed, whereas in OWL 2 we need only one axiom to do by using owl:AllDisjointClasses constructor. In OWL 2 we can also use just owl:disjointUnionOf to define a class as a union of other classes, all of which are pairwise disjoint. Another issue is that sometimes we want to state that two individuals are not related by a property, or to assert values that an individual does not have, OWL 2 provides a means to do that by using owl:NegativePropertyAssertion construct.

OWL 2 also adds more constructs for properties. Apparently, OWL constructs were defined based on DL $\mathcal{SHOIN}$, this led to major drawbacks regarding an expressiveness of property declaration as said earlier. OWL 2, however, was designed based on DL $\mathcal{SROIQ}$ [22] by including new constructs for expressing additional restriction on properties, new characteristics of properties, and incompatibility of properties. These new constructs significantly enhance the expressiveness of the language.

For example, some constructors were added to OWL 2 for declaring that properties are reflexive, irreflexive, or asymmetric. owl:asymmetricProperty defines a property is asymmetric, i.e., if A is related to B via this property, then B is never related to A via this rule. Reflexive property is defined by owl:reflexive, meaning that every individual A is related to itself via this rule. Using reflexive property OWL 2 defines Self Concept, which is a set of individuals related with themselves via a particular property. In addition, irreflexive property is defined by owl:irreflexive.

Moreover, to increase OWL 2's relational expressiveness, some constructs for defining relationships among properties are introduced. OWL 2 provides owl:propertyDisjointWith and owl:AllDisjointProperties to declare disjoint properties. OWL 2 also provides owl:propertyChainAxiom to express property chains, i.e. propagation of one property to another. Table 3.1 briefs some differences between OWL and OWL 2.

Features	OWL	OWL 2
Description logic	based on DL $\mathcal{SHOIN}$	based on DL $\mathcal{SROIQ}$
Disjoint Classes	owl:disjointWith: two classes are disjoint	owl:AllDisjointClasses: several classes are pairwise disjoint.
Property characteristics	transitive, symmetric, functional	Extra-property: asymmetric, reflexive, irrefexive
Property chains	not support	owl:propertyChainAxiom
Negated property assertion	not support	owl:NegativePropertyAssertion
Disjoint Properties	not support	owl:propertyDisjointWith, owl:AllDisjointProperties
Self Concept	no-suport	owl.Self

Table 3.1 Comparison between OWL and OWL 2

3.2.3 Expressing Rules in OWL 2

Some previous SW ontology languages such as DAML+OIL and OWL were developed based on DL $\mathcal{SHOIQ}$. $\mathcal{SHOIQ}$ provides a variety of constructors for building class expressions. The DL class expressions can be demonstrated as being corresponded to first order logic (FOL) which is used to formulate rules. Table 3.2 shows the FOL formulae correspond to the DL class expressions.

According to this correspondence, an FOL sentence can be expressed in DL as well as in DAML+OIL or OWL. For example, a rule of the form: $C(x) \wedge \neg D(x) \rightarrow E(x) \vee F(x)$ can be rewritten as a DL axiom: $C \sqcap \neg D \sqsubseteq E \sqcup F$, and a rule of the form: $C(x) \wedge R(x,y) \rightarrow E(x)$ can be rewritten in DL as the axiom: $C \sqcap R.\top \sqsubseteq E$. However, with some rules such as:

(A) $\mathtt{hasParent}(x,y) \wedge \mathtt{hasBrother}(y,z) \rightarrow \mathtt{hasUncle}(x,z)$, and

(B) $\mathtt{Man}(x) \wedge \mathtt{hasChild}(x,y) \rightarrow \mathtt{fatherOf}(x,y)$.

There is no correspondence so they cannot be rewritten as DL axioms, however to be able to do so we need some extra axioms in DL $\mathcal{SROIQ}$, these are the new features in OWL 2 which we introduced in previous section.

Expression	DL	FOL
subclassOf	$C \sqsubseteq D$	$\forall x.C(x) \rightarrow D(x)$
subpropertyOf	$P_1 \sqsubseteq P_2$	$\forall x,y.P_1(x,y) \rightarrow P_2(x,y)$
transitiveProperty	$P^+ \sqsubseteq P$	$\forall x,y,z.(P(x,y) \wedge P(y,z)) \rightarrow P(x,z)$
functionalPropery	$\top \leq 1\, P$	$\forall x,y,z.(P(x,y) \wedge P(x,z)) \rightarrow y=z$
inverseProperty	$P \equiv Q^-$	$\forall x,y.P(x,y) \rightarrow Q(y,x)$
intersectionOf	$C_1 \sqcap ... \sqcap C_n$	$C_1(x) \wedge ... \wedge C_n(x)$
unionOf	$C_1 \sqcup ... \sqcup C_n$	$C_1(x) \vee ... \vee C_n(x)$
complementOf	$\neg C$	$\neg C(x)$

Table 3.2 DL FOL equivalence

OWL 2, which is based on the more expressive DL $\mathcal{SROIQ}$ [22], supports the Role Inclusion Axiom (RIA) or so-called the "property chain" axiom and the Self Concept, and these can be used to express more forms of rules, including that in the previous example.

Role Inclusion Axioms are constructs of the form $R \circ S \sqsubseteq T$ where $\circ$ is a binary composition operator. This form is equivalent to an FOL formula: $\forall x,y,z.(R(x,y) \wedge S(y,z)) \rightarrow T(x,z)$. By adopting this, rule A can be easily rewritten as a DL $\mathcal{SROIQ}$ axioms:

$$\texttt{hasParent} \circ \texttt{hasBrother} \sqsubseteq \texttt{hasUncle}.$$

The Self Concept allows one to express a "local reflexive" property, e.g. $R(x,x)$, in which a role R relates an individual x to itself. The Self Concept can be used to transform a property $R(x,x)$ into a class C_R and vice versa, and this is due to a DL $\mathcal{SROIQ}$ axiom $C_R \equiv \exists R.\texttt{Self}$. Therefore to derive the DL $\mathcal{SROIQ}$ axioms that corresponding to rule B, we first transform a class Man into a property P_{Man} by introducing an axiom $\texttt{Man} \equiv \exists P_{Man}.\texttt{Self}$, we then apply the previous RIA. As a result, rule B will be equivalent to these DL $\mathcal{SROIQ}$ axioms:

$$\texttt{Man} \equiv \exists P_{Man}.\texttt{Self}.$$
$$P_{Man} \circ \texttt{hasChild} \sqsubseteq \texttt{fatherOf}.$$

With these new features, we can express a rather wide range form of rules, if they satisfy certain restrictions [17], in the form of OWL 2 axioms. Our contribution in this thesis is to provide a reasoning framework to deal with new features in OWL 2, so that it can reason with ontologies and rules express in OWL 2.

3.3 Conclusion

We have explained two SW languages SWRL and OWL 2 for expressing SW ontologies with rules. We will adopt these two SW languages in our framework to explain in chapter 5, 6.

Chapter 4

Meta-logical Framework for Agent Communication of Semantic Web Information

A Meta-logical Framework for Agent Communication of Semantic Web Information (or briefly MAC-SWI framework) was proposed by Hirankitti & Tran [1, 2, 3, 4] is multi-agent system for reasoning with SW ontologies expressed by OWL. In thesis we extend this this framework so that it can reason with SW ontologies with rules expressed by SWRL and OWL 2. This chapter we describe this framework and then in the next two chapters we present our framework.

4.1 MAC-SWI Framework

MAC-SWI is a meta-logical framework for agent communication of SW information, expressed in a meta-logic. Figure 4.1 shows the architecture of MAC-SWI framework, each block in this figure represents one agent which includes three parts: meta-programs of SW ontologies, a meta-interpreter, and the communication facility. Each of the meta-programs contains a meta-level description of an SW ontology. That is, an ontology expressed in its native form, e.g. in RDF, RDF Schema, and OWL languages, is transformed into a meta-logical form. The meta-interpreter is the system's inference engine which is used to infer implicit and explicit information from the meta-logical representation of the ontologies. The communication facility supports communication among individual agents. Our enhancement in this thesis focuses on processing SW information by one single agent. Figure 4.2 depicts components of a single agent.

In next sections we will review the components of one single agent of MAC-SWI framework, we first review meta-languages and meta-programs for SW ontology, and we then recall the definition of meta-interpreter.

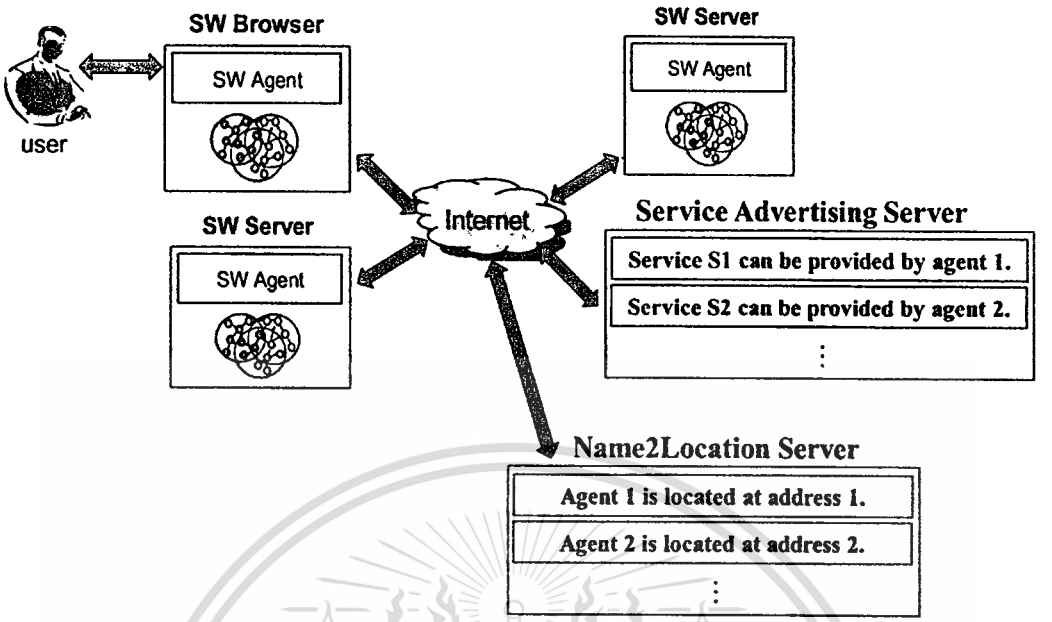


Figure 4.1 MAC-SWI Framework

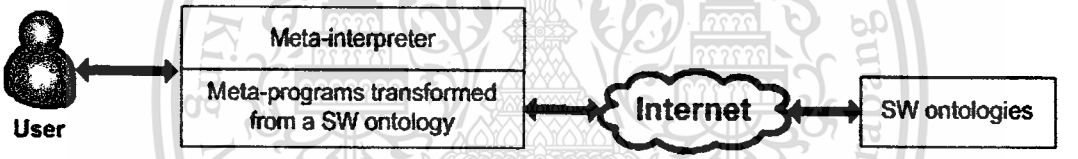


Figure 4.2 A single agent

4.1.1 Object language

Object language is used to describe a domain of discourse, it specifies objects and their relationships at the real world. For example, *John goes to school* is a proposition stating that the person John does the action *go to school*. But if we refer to "John", this is a name of John, containing four letters. Also "*John*" is a noun, "*go*" is a verb, "*to*" is a proposition, and "*school*" is another noun. So "*John*" is a syntactic representation of *John* and it is a meta-language of the object language *John*.

4.1.2 Meta-language

A meta-language is a language that is used to discuss or describe an object language. In other words a meta-language is a language used to make statements about statements in the object language. Meta-language is used to discuss about another language (the object language).

When a meta-language statement describes property of an object language statement, there is an important syntactic relationship between the two languages: the object language statement acts as a term in the meta-language statement.

For example, with the object language statement *John goes to school*, at meta-language '*John*' is a string with four letters used to name the person. This string also represents the subject of the sentence.

In our framework we defined meta-languages to describe Semantic Web Ontology Language. These meta-languages are defined in the following section.

4.1.3 Meta-languages of the Semantic Web Ontology

SW ontologies are created by using SW markup languages: RDF, RDFS, and OWL. In this form, an ontology is sufficiently to be exchanged among applications in the web environment, but it is difficult and not efficient when an application processes data in an ontology.

In MAC-SWI, in order to support an agent to reason with SW ontologies efficiently, logic programming approach is used to represent and reason with SW ontologies. According to this approach, two meta-languages for expressing information expressed in an SW ontology are defined. Each meta-language is used for different kinds of information contained in an SW ontology.

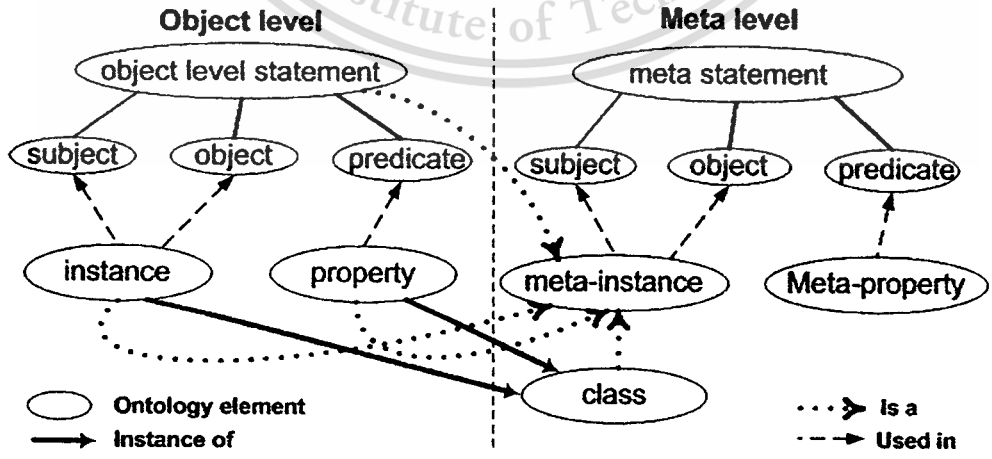


Figure 4.3 Ontology elements at object level and meta level

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 depicted in Figure 4.3. At the object level, an instance can be an individual or a literal of a domain, e.g. `'john'`, and property is a relationship between individuals, or is an individual's attribute, e.g. `'hasSon'`, `'type'`. At the meta-level, a meta-instance can be an individual, a property, a class, or an object-level statement. A meta-property is a property to describe a meta-instance's attribute or a relationship between/among meta-instances, e.g. `'reflexive'`, `'disjointWith'`.

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>`, such as `'f':'son'`, `'f':'hasSon'`, `'owl':'reflexive'`, etc. This qualified name is used throughout this thesis.

In order to describe information at these two levels, two meta-languages were developed: the meta-language for object information (ML) and the meta-language for meta-information (MML). An object language specifies objects and their relationships in the real world while a meta-language describes the syntactic form of the object language. For an SW ontology, ML is used to describe instances and their relationships, whereas MML is used to describe classes, class instances, properties and their relationships.

- **Meta-language for object level (ML)**

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

Meta-constant specifies a name of an object 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 totally separated.

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

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

Meta-term is either a meta-constant or a meta-variable or meta-function symbol applied to a tuple of meta-terms, e.g. `'f':'hasSon', 'owl':'reflexive'`. To express 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 variable, e.g. `'f':'hasSon'('f':'fa','f':'son')`. The meta-term expressing an *object-level sentence* is a logical-connective function symbol applied to the tuple of these terms, e.g., `'f':'hasSon'('f':'fa','f':'son') '←' true`.

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':'hasSon'('f':'fa','f':'son')←true).
```

- **Meta-language for meta-level (MML)**

Apart from the object language, an SW ontology also defines classes, properties, their relationships, as well as class-instance relations. This information is precisely *meta-information of the object-level*. MML is defined to describe such information. MML language 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 set operator applied on classes such as union, intersection, and complement; or a meta-predicate name being a name of a relation between entities; or a name of characteristic of a property, which may fall into one of the following categories:

Class-class relations: 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 take these relations as attributes of instances, of classes, or of properties, e.g. comment, label.

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

Meta-term being either a meta-constant or a meta-variable or meta-function symbol applied to a tuple of meta-terms, e.g. `equivalentClass(C,EC)`, `Man(son)`, `fatherOf(fa,son)`. A name of a class, a property, etc., can be referenced by a meta-term in the forms of: `namespace:name` e.g. `'f':'Man'`. `'f':'fatherOf'`.

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':'propertyDisjointWith'(P,DP) ←
        'owl':'propertyDisjointWith'(DP,P).
```

Meta-statement being a meta-predicate or meta-predicates connected by logical connective. It has two forms meta_statement(meta-level-sentence) and axiom(meta-level-sentence), the latter represents a rule for a mathematical axiom, e.g.:

```
meta_statement('owl':'propertyDisjointWith'
      ('f':'likes','f':'dislikes') ← true).

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

The second rule represents an inference 'axiom' used for supporting reasoning process of an inference engine.

4.1.4 Meta-programs of the Semantic Web Ontology

Each SW ontology is transformed into a meta-program containing a (sub-)meta-program expressed in **ML**, called **MP**, and a (sub-)meta-program expressed in **MML**, called **MMP**. Another meta-program expresses some mathematical axioms for classes and properties called **AMP** is also needed for the inference engine to reason with **MP** and **MMP**.

- **Meta-program for the object-level (MP)**

MP contains meta-statements expressing information of instances and their relationships which are described in SW ontology by OWL language. Information in an SW ontology is expressed by a set of RDF statements. Each RDF statement is a triple of subject, predicate, and object. In this framework an RDF statement in an original ontology is transformed into a logical statement, that is, an RDF triple of the form

(Subject,Predicate,Object) is transformed into Predicate(Subject,Object) or shortly P(S,O). Given this form of a RDF statements, meta-statements in **MP** are in the form of: statement(P(S,O) ← true).

Here is an example of **MP**:

```
statement('f':'hasParent'('f':'son','f':'fa') ← true).
statement('f':'hasAge'('f':'son',18) ← true).
```

- **Meta-program for the object-level (MP)**

MMP contains meta-statements for classes, properties, their relationships, and class-instance relations in terms of meta-rules. The **MMP** is represented in the following forms:

```
meta_statement(P(S,O) ← true),
meta_statement(P(S,Os) ← true), and
meta_statement(C(Prop) ← true),
```

where P, S, O are predicate, subject, and object of a triple (S, P, O) defined in the ontology. C is a characteristic of a property Prop. Os is a tuple composing of several objects.

Here are some tipycal statements 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).
```
//the classes C and SC are equivalent

```
meta-statement('owl':'disjointWith'(C,DC) ← true).
```
//the classes C and DC are disjoint

```
meta-statement('rdf':'type'(I,C) ← true).
```
//the instance I is an instance of the class C

```
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 intersection of classes in Cs

...

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,SP) ← true).
```
//the properties P and SP are equivalent

```
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

```
meta-statement('owl':'inverseOf'(P,IP) ← true).
```
//the property P is inversion of the property IP

```
meta-statement('rdfs':'transitive'(P) ← true).
```
//the property P is transitive

```
meta-statement('rdfs':'symmetric'(P) ← true).
```
//the property P is symmetric

```
meta-statement('rdfs':'functional'(P) ← true).
```
//the property P is functional

- **Meta-program for the Axioms (AMP)**

The **AMP** program contains mathematical and inference axioms for classes, instances, and properties. The axioms are expressed in the meta-rule form as follows:

```
axiom(P(S,O) ← Body).
```

The **AMP** contains a complete set of axioms which reflects almost inference rules for OWL. Each axiom is used for a relationship between two classes, two properties, an instance and its class. There are some typical axioms of the **AMP** program:

//the following relations of classes and properties are transitive

```
axiom('owl':'equivalentClass'(C,EC) ←                          (atec)
    'owl':'equivalentClass'(C,C1) ∧
    'owl':'equivalentClass'(C1,EC)).

axiom('rdfs':'subClassOf'(C,SC) ←                              (atsc)
      'rdfs':'subClassOf'(C,C1) ∧ 'rdfs':'subClassOf'(C1,EC)).

axiom('owl':'equivalentProperty'(P,EP) ←                       (atep)
      'owl':'equivalentProperty'(P,EP1) ∧
      'owl':'equivalentProperty'(EP1,EP)).

axiom('rdfs':'subPropertyOf'(P,SP) ←                           (atsp)
'rdfs':'subPropertyOf'(P,P1) ∧ 'rdfs':'subPropertyOf'(P1,SP)).

axiom('owl':'sameAs'(I,SI) ←                                   (atsa)
    'owl':'sameAs'(I,SI1) ∧  'owl':'sameAs'(SI1,SI)).
...
```

//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':'inverseOf'(P,IP) ← 'owl':'inverseOf'(IP,P)  (asip)
...
```

//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) ← 'owl':'inverseOf'(IP,P) ∧ IP(O,S)).       (acip)
axiom(P(S,O) ← 'rdfs':'subPropertyOf'(SP,P) ∧ SP(S,O)).  (acsp)
axiom(P(S,O) ← 'owl':'transitive'(P) ∧ P(S,O1) ∧ P(O1,O)). (actp)
axiom(P(S,O) ← 'rdfs':'symmetric'(P) ∧ P(O,S)).          (acsmp)
...
```

4.1.5 The Semantic Web Meta-interpreter

A meta-interpreter is used to manipulate and reason with the meta-programs (MP, MMP, and AMP) which are translated from SW ontology. It can also be used to develop an intelligent agent to reason with SW ontologies. In MAC-SWI framework, the meta-interpreter is defined by a demo predicate of the form `demo(A)`. With this predicate the meta-interpreter can infer an answer A from the meta-programs. The interpreter is defined by adapting the Vanilla meta-interpreter [26] for reasoning with the meta-programs, which 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, and (3) `axiom(A←B)` for a supporting mathematical axiom. The definition of `demo/1` is:

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

The first clause (true) is the basic case for proving an atom true. The second clause (conj) is used for proving a conjunction goal. Three last clauses (ost), (mst), and (ast) are used for proving three meta statements from the three meta-programs **MP**, **MMP**, and **AMP** respectively.

4.1.6 The Reasoning of SW Meta-interpreter

The reasoning of the SW meta-interpreter is described as follows. The constant symbol `true` is evaluated to true. To prove that the conjunction `A'∧'B` is true, the meta-interpreter must prove `A` is true and `B` is true. To prove a goal `A` be true, there must exist either a `statement(A'←'B)`, or `meta_statement(A'←'B)`, or `axiom(A'←'B)` in meta-programs translated from an SW ontology and `B` must be proven to be true.

To illustrate the reasoning of the SW meta-interpreter we consider the following example of meta-program transformed from an SW ontology. Suppose we have an OWL assertion in the ontology saying that a person 'fa' is father of peson 'son'. This assertion is translated into a meta-statement in meta-program MP as follows.

```
statement('f':'isFatherOf'('f':'fa', 'f':'son')←true).
```

The tracing of the deduction process by the SW meta-interpreter for answering the query `demo('f':'isFatherOf'('f':'fa',X))` from the meta-program is shown in Figure 4.4

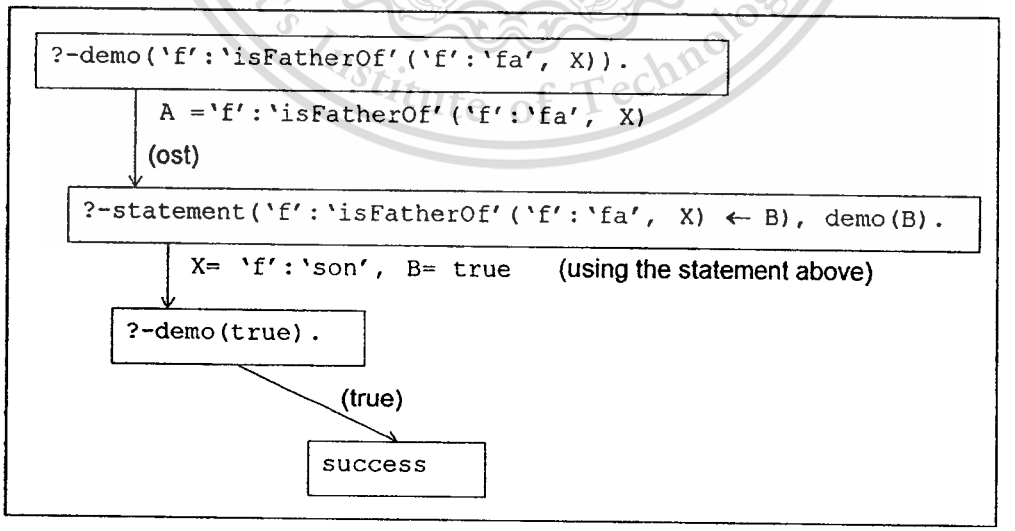


Figure 4.4 Tracing the deduction process of SW meta-interpreter

4.2 Our Extended Framework for Reasoning with SW Ontologies with Rules

As we said in previous chapter, adding rules to SW ontologies has been become an important issue in SW research, and many proposals have been proposed in order to extend SW ontologies with rules, such as some research works we have surveyed in chapter 3. A single agent in MAC-SWI framework was developed with the concentration to work with ontologies described in RDFS, and OWL which have not supported for rules in SW ontologies.

Our proposal in this thesis is to enhance the MAC-SWI framework so that it can handle with the new markup language for SW ontologies with rules. In this thesis we focus on improving the framework so that it can work with SWRL and OWL 2. With the new improvement of the meta-logical knowledge representation, SW ontologies and rules expressed by SWRL and OWL 2 will be transformed to the meta-programs. We also slightly modify the meta-interpreter and add new extra auxiliary axioms in order to support reasoning with SW ontologies and rules. The further details of this framework will be described in the following chapters.

4.3 Conclusion

This chapter reviews the MAC-SWI framework, which we investigate. We have studied it carefully on processing SW information by one single agent in this framework and have found that while the framework work well will SW ontologies expressed in OWL, it has to be extended in order to be able to reason with ontologies and rules expressed in new SW markup languages, SWRL, OWL 2. This led us to do an enhancement on the meta-logical representation for SWRL and OWL 2, so that the MAC-SWI framework can work with them. In chapter 5 we present an extension to the MAC-SWI framework to deal with SWRL ontologies; and in chapter 6 we present an extension of the framework to deal with ontologies with rules in OWL 2.

Chapter 5

A Meta-logical Framework for Reasoning with SWRL Ontologies

5.1 Overview the Framework

In this thesis the MAC-SWI framework, which was designed to support reasoning with OWL ontologies, is enhanced with ability to reason with Semantic Web Ontologies and rules expressed in SWRL. Our meta-logical framework can simply be illustrated in Figure 5.1.

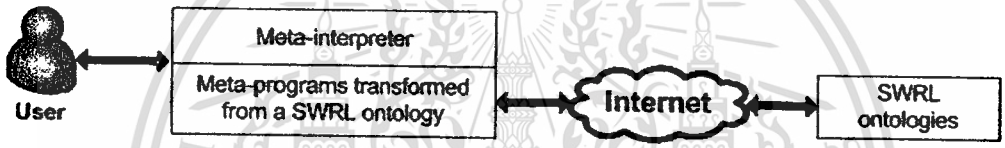


Figure 5.1 Meta-logical framework for reasoning with SWRL ontologies

Our framework is defined as a tuple of ⟨ *meta-programs of SWRL ontologies, meta-interpreter* ⟩. The former is in the form of logical sentences representing a meta-level description of SW ontologies with rules. That is, the SW ontologies and rule described by SWRL are transformed into meta-logical representations. The latter is a meta-interpreter, in the form of a demo (meta-)program, which is used to infer explicit as well as implicit information, or in other words draw conclusions, from the former.

In order to support rules, which are expressed by SWRL, the meta-languages in the MAC-SWI framework need to be modified with the support for syntax of rules and the meta-program in the MAC-SWI framework has to be extended with some new forms of meta-statements.

To explain our framework, in the following sections we first introduce meta-representation for SW ontologies with rules expressed in SWRL, we then explain the meta-programs in details, and add more auxiliary axioms to the meta-programs. Finally we present our meta-interpreter which is our inference engine for reasoning with SW ontologies with rules.

5.2 Meta-representation of SWRL ontologies

In SWRL ontology, a rule is in form of Horn-clause in the RuleML syntax. A rule describes relationship between instances of OWL classes, or between instances and literals, or between instances and classes. As we distinguish between object-level information describing instances, their relationships and meta-level information describing classes, properties and their relationships of an ontology, SWRL rule can be used to express information of both levels. SWRL rules, therefore, will appear in both Meta-language for object information (ML) and Meta-language for meta-information (MML).

Besides, SWRL supports a set of built-in functions to use for processing datatypes such as strings and numbers. In order to cope with this, in our framework SWRL built-in functions will be mapped to corresponding built-in predicates in Prolog.

Since the MAC-SWI framework lays down a foundation of our framework most of the meta-language elements will be adopted in our framework, so the part that is borrowed from the MAC-SWI framework will be put in a block without a background color, but any part of the meta-languages that is different from that of the MAC-SWI framework will be put in a block with a gray color.

5.2.1 Meta-languages for an SWRL ontology

- **Meta-language for Object Information of an SWRL ontology (ML)**

Objects, their relationships as well as rules at the object level are specified in an SWRL ontology and this information is expressed by the elements of ML below.

Meta-constant specifies a name of an object and a literal, e.g. `'son'`, including a reference, e.g. a namespace, the latter is a meta-constant of MML. This means that ML and MML are not totally 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 objects, or a name of an object's property—i.e. an object-level predicate name, such as `'hasSon'`, `'name'`. It also stands for other meta-level function symbol, e.g. `'←'`, `'∧'`, `':'`.

Meta-term is either a meta-constant or a meta-variable or meta-function symbol applied to a tuple of meta-terms, e.g. `'f':'hasSon'`, `'owl':'reflexive'`. To express 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 variable, e.g. `'f':'hasSon'('f':'fa','f':'son')`. 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-variable appearing in the object-level sentence are universally quantified. One form of this sentence is a Horn-clause, e.g.

```
'f':'hasFather'(Ch,F) '←'
        'f':'hasParent'(Ch,F) '∧' 'rdf':'type'(F,'f':'Man').
```

The meta-term, expressing an object-level predicate, is equivalent to a form of Horn-clause with an empty body. Thus, we can put `true` instead of the emptiness in its body, e.g.

```
'f':'hasSon'('f':'fa','f':'son') '←' true.
```

Meta-statement for the object level reflects an object-level sentence to its existence at the meta-level. It has a form *statement(object-level-sentence)*, note that this statement is used to represent a Horn-clause rule translated from an SWRL rule, e.g.

```
statement('f':'hasSon'('f':'fa','f':'son') '←' true).
statement('f':'hasFather'(Ch,F) '←'
    'f':'hasParent'(Ch,F) '∧' 'rdf':'type'(F,'f':'Man')).
```

- **Meta-language for Meta-Information of an SWRL ontology (MML)**

In our meta-logical framework, meta-information of an SWRL ontology, which is classes, properties, their relationships, class-instance relations, as well as SWRL built-in functions, is expressed by the **MML** language. The components of **MML** consist of meta-constant, meta-variable, meta-function symbol, meta-term and meta-statement which are defined as follows.

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 `':'`; or a set operator applied on classes such as union; or a meta-predicate name being a name of a relation between entities; or a name of characteristic of a property, which may fall into one of the following categories:

- *Class-class relations*: equivalent class of, disjoint with, etc.
- *Class-instance relations*: instance of, class of, etc.
- *Property-property relations*: subproperty of, chain of, etc.

- *Relations between literals and instances/classes/properties*: we can take these relations as attributes of instances, of classes, or of properties, e.g. comment, label.
- *Characteristics of properties*: reflexive, asymmetric, etc.

Meta-term is either a meta-constant or a meta-variable or meta-function symbol applied to a tuple of meta-terms e.g. `equivalentClass(C,EC)`, `Man(son)`, `fatherOf(fa,son)`, in our framework, a name of a class, a property, etc., can be referenced by a meta-term in the forms of: `namespace:name` e.g. `'f':'Man'`, `'f':'fatherOf'`, `'owl':'equivalentClass'`.

When a meta-term expresses a meta-level predicate stating a relation between entities, it has the form `Pred(Sub,Obj)`. When it expresses a meta-level predicate stating a characteristic of a property, it has the form `Pred(Prop)`, and when it express a built-in function, it has the form `builtinAtom(Fc,`Arg_1`,...`Arg_n`)` where `Pred` is a meta-predicate name, `Fc` is a built-in function name, `Sub`, Arg_1, ...Arg_n 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. We treat the sentence as a Horn-clause rule, for the empty tuple in the body we put `true` there instead, e.g.

```
'rdf':'type'('f':'M02','f':'Man') '←' true.
'owl':'propertyDisjointWith'(P,DP) '←'
        'owl':'propertyDisjointWith'(DP,P).
```

Meta-level sentence expresses a built-in function has a form as follows:

`builtinAtom(Fc,`Arg_1`,...`Arg_n`)` `'←'` `builtin(FcinPrlog).`

where `Fc` is name of built-in function in SWRL, and `FcinPrlog` is a corresponding built-in predicate in Prolog of `Fc` with arguments are Arg_1, ...Arg_n For example built-in `'lessthan'` in SWRL has a corresponding Prolog built-in predicate is <.

Meta-statement being a meta-representation of a meta-level sentence accessible by our meta-interpreter. It has two forms *meta_statement(meta-level-sentence)* and *axiom(meta-level-sentence)*, the latter presents a rule for a mathematical axiom, and a built-in atom in SWRL, e.g.

```
meta_statement('rdf':'type'('f':'M02','f':'Man') '←' true).
axiom('owl':'propertyDisjointWith'(P,DP) '←'
            'owl':'propertyDisjointWith'(DP,P)).
axiom('swrlb':'builtinAtom'('lessThan',x,y) '←' builtin(x < y)).
```

5.2.2 Meta-programs of an SWRL ontology

To formulate meta-programs from SW ontologies to use in our framework, each SWRL ontology is transformed into a meta-program containing a (sub-)meta-program expressed in **ML**, called **MP**, and a (sub-)meta-program expressed in **MML**, called **MMP**. Another meta-program expresses some mathematical axioms for classes and properties or expresses some SWRL built-in atoms in **MML** called **AMP** is also needed for the inference engine, i.e. our meta-interpreter, to reason with **MP** and **MMP**.

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

MP contains information about instances and their relationships in terms of meta-statements for the object level: `statement(P(S,O) ← true)`, `statement(P(S,O) ← Body)`, where `Body` is either single object-level predicate or conjunction of object-level predicates. The latter form expresses a Horn-clause rule which is described by SWRL. That is, a rule encoded in SWRL is transformed to a meta-statement with the head is in the form of `P(S,O)` and the body has the form of conjunctions of object-level predicates which are directly extracted from the original rule in SWRL.

To solve a goal that matches with the head `P(S,O)` of the meta-statement `statement(P(S,O) ← Body)`, the meta-interpreter recursively solves the `Body` of this meta-statement and the goal `P(S,O)` is evaluated *'true'* if the `Body` is proved to be *'true'*.

For example, we want to express a rule R1 saying that "if P is a parent of Ch and U is a brother of P, then U is the uncle of Ch". The SWRL syntax of this rule is defined as in Figure 3.1, the corresponding meta-statement of this rule is defined as follows in Figure 5.2.

```
statement('eg':'hasUncle'(Ch,U) ←
      'eg':'hasParent'(Ch,P) ∧ 'eg':'hasBrother'(P,U)).
```

Figure 5.2 An object-level meta-statement for a SWRL rule example

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

To express an SWRL rule with antecedent `P(S,O)` which is information at meta-level, in this case it is instance-class relationship, we add a meta-statement with the form `meta_statement(P(S,O) ← Body)` to the **MMP** program. Similarly, to prove the head

P(S,O) of the meta-statement meta_statement(P(S,O) ← Body) is *'true'*, the meta-interpreter has to recursively solve the Body of this meta-statement and prove it is *'true'*.

Here we show an example of this kind of SWRL rule, a rule R2 saying that "The person P is an instance of class Adult, if person P has age is A and the age A is greater than or equal 18". This rule is expressed by SWRL syntax in Figure 5.3 and can be transformed to a meta-statement at meta-level as in Figure 5.4.

```
<swrl:Imp rdf:ID="Def-Adult">

  <swrl:body rdf:parseType="Collection">
    <swrl:classAtom>
      <swrl:classPredicate rdf:resource="&eg;Human"/>
      <swrl:argument1 rdf:resource="#p" />
    </swrl:classPredicate>
    <swrl:individualPropertyAtom>
      <swrl:propertyPredicate rdf:resource="&eg;hasAge"/>
      <swrl:argument1 rdf:resource="#p" />
      <swrl:argument2 rdf:resource="#a" />
    </swrl:individualPropertyAtom>
    <swrlx:builtinAtom swrlx:builtin="&swrlb;#greaterThanOrEqual">
      <swrl:argument1 rdf:resource="#a" />
      <owl:DataValue owl:datatype="&xsd;#int">18</owl:DataValue>
    </swrlx:builtinAtom>
  </swrl:body>

  <swrl:head rdf:parseType="Collection">
    <swrl:classAtom>
      <swrl:classPredicate rdf:resource="&eg;Adult"/>
      <swrl:argument1 rdf:resource="#p" />
    </swrl:classPredicate>
  </swrl:head>

</swrl:Imp>
```

Figure 5.3 An SWRL rule with head is information at meta-level

```
meta-statement('rdf':'type'(P,'eg':'Adult') ←
     'rdf':'Type'(P,'eg':'Human') ∧ 'eg':'hasAage'(P,A) ∧
      'swrlb':'builtAtom'('greaterThanOrEqual',A,18)).
```

Figure 5.4 A meta-level meta-statement for a SWRL rule example

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

AMP contains mathematical and inference axioms for classes, instances and properties. In other words **AMP** is set of statements which formulate the OWL primitives (OWL semantic). In addition, **AMP** also contains axioms for SWRL built-in functions; the purpose of introducing these axioms is to provide a way to translate SWRL built-in atoms in into the corresponding

Prolog atoms with matched built-in predicates, so that later our meta-interpreter can work with the SWRL built-in functions.

Many axioms from MAC-SWI framework were borrowd to support OWL primitives such as the axioms related to equivalent classes, the axioms related equivalent properties, the axioms related to characteristics of a property, etc., as those given in chapter 4. In our framework we add some new axioms to support SWRL built-ins [24]. To do that we study the function of each SWRL built-in, and then find or create a Prolog predicate with the same function as the SWRL built-in. For instance, with SWRL built-in swrlb:lessThanOrEqual, the corresponding Prolog predicate of it is '=<'. Table 5.1 shows some typical SWRL built-ins and their matched Prolog built-in predicates.

SWRL built-in function	Prolog built-in predicate
swrlb:equal	=:=
swrlb:notEqual	=\=
swrlb:lessThan	<
swrlb:lessThanOrEqual	=<
swrlb:greaterThan	>
swrlb:greaterThanOrEqual	>=
swrlb:add(Z,X,Y)	Z is X+Y
swrlb:subtract(Z,X,Y)	Z is X-Y
swrlb:multiply(Z,X,Y)	Z is X * Y
swrlb:divide(Z,X,Y)	Z is X/Y
swrlb:stringConcat(P,W,J)	cat(P,W,J)
swrlb:stringLength(S,N)	len(S,N)
swrlb:length(L,N)	length(L,N)

Table 5.1 SWRL built-ins and the corresponding Prolog predicates

Here are examples of axioms for SWRL built-in functions in **AMP**:

```
axiom('swrlb':'builtinAtom'('lessThan',x,y) ← builtin(x < y)).(albia)
axiom('swrlb':'builtinAtom'('equal',x,y) ← builtin(x = y)).   (aebia)
axiom('swrlb':'builtinAtom'('lessthanOrEqual',x,y) ←          (alebia)
                            builtin(x =< y)).
axiom('swrlb':'builtinAtom'('greaterThan',x,y) ←              (agbia)
                            builtin(x > y)).
axiom('swrlb':'builtinAtom'('greaterThanOrEqual',x,y) ←       (agebia)
                            builtin(x >= y)).
...
```

5.3 A Meta-Interpreter for Reasoning with the Meta-programs

In our framework SW ontologies and rules are transformed into meta-programs. The exploitation of information defined in those ontologies and rules is a deduction process to derive conclusion from the meta-programs for some queries. A meta-interpreter is defined for this purpose. Our meta-interpreter is defined by adapting the Vanilla-interpreter [26] which was introduced in logic programming.

In our framework, there are three types of meta-programs **MP**, **MMP**, **and AMP** which are used to expressed information of SW ontologies and rules at the meta-level. **MP** is a set of meta-level statements in the form of `statement(A ← B)` for object-level information, **MMP** is a set of meta-level statements in the form of `meta_statement(A ← B)` for meta-information, and **AMP** is a set of meta-level statements with the form of `axiom(A ← B)` for mathematical axioms to support reasoning with SWRL ontologies. The Vanilla meta-interpreter in our framework is modified in order to handle with these three types of meta-level statements. It is defined by the meta-predicate `demo/1` as follows.

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

The first clause (true) is the basic for proving an atom `true`. The second clause (conj) is used for proving a conjunctive goal. The three clauses (ost), (mst), and (ast) are used for proving a goal which is supported by one of the three meta statements `statement(A ← B)`, `meta_statement(A ← B)`, or `axiom(A ← B)` of the three meta-programs **MP**, **MMP**, and **AMP** respectively.

In addition, our SW meta-interpreter can also work with SWRL built-in functions via their translation to their corresponding Prolog atoms with matched built-in predicates. The last clause (btin) is defined to handle it.

5.4 Query Answering with SWRL Ontology in Our Framework

We use the family ontology taken from [40] for a demonstration purpose of query answering by our meta-interpreter. This SWRL ontology describes the usual classes, e.g. `Peson`, `Man`, `Woman`, etc., relationships, e.g. `hasParent`, `hasChild`, `hasFather`, etc., and the rules about `hasUncle`, `hasSon`, `hasBrother`, etc., in content of a family. In this family ontology some explicit information about a person is easily be represented. However, many implicit information need to be derived by a deductive process. This deductive process in our framework is performed as follows. Firstly, the SWRL ontology is transformed into meta-programs, Figure 5.5 and Figure 5.6 depict some parts of **MP** meta-program, and **MMP** program. Our meta-interpreter then, with the support of **AMP** program, reasons with the **MP**, and **MMP** meta-programs to derive answers for the queries. Figure 5.7 shows examples of query answering upon the family ontology and give hustification for each answer.

- **The MP program**

```
statement('f':'hasParent'('f':'M02','f':'M01') ← true).        (a1)
statement('f':'hasParent'('f':'M02','f':'F01') ← true).        (a2)
statement('f':'hasParent'('f':'M03','f':'M02') ← true).        (a3)
statement('f':'hasParent'('f': 'M05','f':'M02') ← true).       (a4)
statement('f':'hasParent'('f':'F02','f':'M05') ← true).        (a5)
statement('f':'hasAge'('f':'M02',25) ← true).                  (a6)
statement('f':'hasAge'('f':'M03',18) ← true).                  (a7)

//Statements expressing SWRL rules of abject information
statement('f':'hasFather'(C,F) ←                               (r1)
    'f':'hasParent'(C,F) ∧ 'rdf':'type'(F,'f':'Man')).
statement('f':'hasMother'(C,M) ←                               (r2)
    'f':'hasParent'(C,M) ∧ 'rdf':'type'(M,'f':'Woman')).
statement('f':'hasSibling'(P1,P2)← 'f':'hasParent'(P1,P3) ∧    (r3)
    'f':'hasParent'(P2,P3) ∧ 'owl':'differentFrom'(P1,P2)).
statement('f':'hasBrother'(P,B) ←                              (r4)
    'f':'hasSibling'(P,B) ∧ 'rdf':'type'(B,'f':'Man')).
statement('f':'hasUncle'(P1,P2) ←                              (r5)
    'f':'hasParent'(P1,P3) ∧ 'f':'hasBrother'(P3,P2)).
statement('f':'hasSon'(P,C) ←                                  (r6)
    'f':'hasChild'(P,C) ∧ 'rdf':'type'(C,'f':'Man')).
```

Figure 5.5 The MP program of the family SWRL ontology

- **The MMP program**

```
meta_statement('owl':'inverseOf'                                    (a1')
     ('f':'hasChild','f':'hasParent') ← true ).

meta_statement('rdf':'type'('f':'M01','f':'Man') ← true).           (a2')

meta_statement('rdf':'type'('f':'M02','f':'Man') ← true).           (a3')

meta_statement('rdf':'type' ('f':'M03','f':'Man') ← true).          (a4')

meta_statement('rdf':'type' ('f':'M05','f':'Man') ← true).          (a5')

meta_statement('rdf':'type'('f':'F01','f':'Woman') ← true).        (a6')


//Statements expressing SWRL rules of meta-information
meta-statement('rdf':'type'(P,'f':'Adult') ←                        (r7)
     'f':'hasAge'(P,A) ∧
     'swrlb':'builtinAtom'('greaterThan',A,18)).

meta-statement('rdf':'type'(P,'f':'Adolescence') ←                  (r8)
     'f':'hasAge'(P,A) ∧
     'swrlb':'builtinAtom'('greaterThan',A,12) ∧
     'swrlb':'builtinAtom'('lessThanOrEqual',A,18)).
```

Figure 5.6 The MMP program of the family SWRL ontology

Figure 5.7 shows query answering by our meta-interpreter:

```
?- demo('f':'hasChild'('f':'M01',X)).
  X = 'f':'M02'.

//The adopted clauses are (acip), ( ast), (a1'), (mst), (a1), (ost), and (true).

?- demo('f':'hasSon'('f':'F01',X)).
     X = 'f':'M02'.

//The adopted clauses are (r6), (ost), (conj), (acip), (ast) , (a1'), (mst), (a2), (a3'), and (true).

?-demo('f':'hasFather'('f':'M02',X)).
     X = 'f':'M01'.

//The adopted clauses are (r1), (ost), (conj), (a1), (a2'), (mst) and (true).

?-demo('f':'hasMother'('f':'M02',X)).
     X = 'f':'F01'.

//The adopted clauses are (r2), (ost), (conj), (a2), (a6'), (mst) and (true).

?-demo('f':'hasBrother'('f':'M03',X)).
     X = 'f':'M05'.

//The adopted clauses are (r3), (r4), (ost), (conj) (a3), (a4), (a5'), (mst) and (true).

?-demo('f':'hasUncle'('f':'F02',X)).
     X = 'f':'M03'.

//The adopted clauses are (r3), (r4), (r5), (ost),(conj), (a5), (a3), (a4), (a4'), (mst),and (true).
```

```
?-demo('rdf':'type'(X,'f':'Adult')).
     X = 'f':'M02'.
```

//The adopted clauses are (r7), (mst), (conj), (a6), (agbia), (ast), (btin), and (true).

```
?-demo('rdf':'type'('f':'M03',X)).
     X = 'f':'Man',
     X = 'f':'Adolescence'.
```

//The 1st answer is supported by (a4'), (mst), and (true)

The 2nd answer is supported by (r8), (mst), (conj), (a7), (agbia), (alebia), (ast), (btin), and (true).

Figure 5.7 Query answering with the family SWRL ontology

5.5 Conclusion

We have presented a meta-logical framework for reasoning with an SWRL ontology. In this paper the MAC-SWI framework that was designed to support OWL, has been extended to accommodate SWRL rules by improving the meta-languages to express Horn-clause rules and modifying the meta-interpreter so that it can work with the newly revised meta-languages.

Chapter 6

A Meta-logical Framework for Reasoning with Ontologies and Rules in OWL 2

In this chapter we propose a framework for reasoning with ontologies and rules expressed in OWL 2. Our meta-logical framework can be simply illustrated in Figure 6.1.

Figure 6.1 Meta-logical framework for reasoning with OWL 2 ontologies

Our framework is defined as a tuple of ⟨ *meta-programs of OWL 2 ontologies, meta-interpreter* ⟩. The former is in the form of logical sentences representing a meta-level description of ontologies with rules expressed in OWL 2. That is, the ontologies and rules expressed in OWL 2 are transformed into meta-logical representations. The latter is a meta-interpreter, in the form of a demo (meta-)program, which is used to infer explicit as well as implicit information, or in other words draw conclusions, from the former. In order to support rules, which are expressed by using the new features in OWL 2, the meta-program given in chapter 4 has to be extended with some new forms of meta-statements.

To explain our framework, in the next sections we first introduce our meta-languages used for formulating the meta-programs of ontologies and rules, then explain the meta-programs in details. Finally we present our meta-interpreter.

6.1 Meta-representation of OWL 2 ontologies

In order to support rules, which are expressed by using the new features in OWL 2, the meta-languages given in chapter 4 is extended with some new forms of meta-statements to express the new features in OWL 2.

6.1.1 Meta-languages for an OWL 2 ontology

OWL 2 adds new constracts to OWL to express the new features. Due to the resemblance between OWL 2 and OWL, for OWL 2 we also define two meta-languages: one for the object-level information in the ontology (ML) and one for the meta-information in the ontology (MML). Almost of meta-language elements are borrowed from the MAC-SWI framework, but some elements are different from these of the MAC-SWI framework. We will highlight these parts in blocks with a gray color.

- **Meta-language for Object-level Information in an OWL 2 Ontology (ML)**

ML is used to describe objects and their relationships of an ontology at object level. The elements of ML are defined as follows.

Meta-constant specifies a name of an object and a literal, e.g. 'son', including a reference, e.g. a namespace, the latter is a meta-constant of **MML**. This means that **ML** and **MML** are not totally 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 objects, or a name of an object's property—i.e. an object-level predicate name, such as `'hasSon'`, `'name'`. It also stands for other meta-level function symbol, e.g. `'←'`, `'^'`, `':'`.

Meta-term is either a meta-constant or a meta-variable or meta-function symbol applied to a tuple of meta-terms, e.g. `'f':'hasSon'`, `'owl':'reflexive'`. To express 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-variable, e.g. `'f':'hasSon'('f':'fa','f':'son')`; for a negative predicate ¬P, it has the form negative(P, S, O).

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':'hasSon'('f':'fa','f':'son')←true).
```

- **Meta-language for Meta-level Information in an OWL 2 Ontology (MML)**

Apart from the object language, an SW ontology also defines classes, properties, their relationships, as well as class-instance relations, and we argue that this information is *meta-information of the object level*. Here we express this information by **MML** which 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. '$\leftarrow$', '$\wedge$', '$\neg$' ('$\neg$' is used to express a classical negation); or ':'; or a name of set operators applied on classes such as union; or a meta-predicate name being a name of a relation between entities; or a name of characteristic of a property, which may fall into one of the following categories:

- *Class-class relations*: equivalent class of, etc.
- *Class-instance relations*: instance of, class of, etc.
- *Property-property relations*: property chain of, etc.
- *Relations between literals and instances/classes/ properties*: we can take these relations as attributes of instances, of classes, or of properties, e.g. comment.
- *Characteristics of properties*: reflexive, asymmetric, etc.

Meta-term being either a meta-constant or a meta-variable or meta-function symbol applied to a tuple of meta-terms. In our framework, a name of a class, a property, etc., can be referenced by a meta-term in the forms of: namespace:name e.g. `'f':'Man'`, `'f':'fatherOf'`, 'owl':'equivalentClass'.

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.

An example of a meta-term expressing a classical negation, is in the form of '$\neg$' `'rdf':'type'(I,C)`, where `I` is a meta-constant specifying an individual, and `C` is a meta-constant specifying a class.

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':'propertyDisjointWith'(P,DP) ←
        'owl':'propertyDisjointWith'(DP,P).
```

Meta-statement being a meta-predicate or meta-predicates connected by logical connective. It has two forms meta_statement(meta-level-sentence) and axiom(meta-level-sentence), the latter represents a rule for a mathematical axiom, e.g.:

```
meta_statement('owl':'propertyDisjointWith'
                ('f':'likes','f':'dislikes') ← true).

axiom('owl':'propertyDisjointWith'(P,DP) ←
        'owl':'propertyDisjointWith'(DP,P)).
```

6.1.2 Meta-programs for an OWL 2 ontology

Similar to our previous framework, an OWL 2 ontology, in this framework, is also translated to three meta-programs **MP**, **MMP** and **AMP** where **MP** is expressed in **ML**, and **MMP** and **AMP** are expressed in **MML**.

To describe new features and rules in an OWL 2 ontology, we define new statements for these three meta-programs.

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

MP contains information about instances and their relationship in the form of meta-statements for the object level: `statement(P(S,O) ← true)`. Here is an example:

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

With the new syntax added to **ML** to describe a negative property assertion, a meta-statement in **MP** for this feature is in the form of `statement(NP(P,S,O) ← true)`, where `NP` expresses a negative property. For example, to declare person M01 is not the father of person M03 we have a meta-statement in **MP** as follows:

```
statement('owl':'negativeProperty'
        ('f':'fatherOf','f':'M01','f':'M03') ← true).
```

With this term form we can define a negative property in the form of '¬'P(S,O), to do that we define an axiom to relate this form to the expression of negative property assertion above. The axiom will be expressed in the meta-program **AMP**.

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

Meta-statements in **MMP** are expressed in three forms as follows:

```
meta_statement(P(S,O) ← true),
meta_statement(P(S,Os) ← true),
meta_statement(C(Prop) ← true),
```

where P, S, O are predicate, subject, and object of a triple (S, P, O) defined in the ontology. C is a characteristic of a property Prop. Os is a tuple composing of several objects. With these three forms the new features in OWL 2 are transformed into **MMP** with the meta-statements shown in Figure 6.2.

New meta-statements for syntactic sugar:

```
meta_statement('owl':'AllDisjointClasses'(Cs) ← true)
// All classes in Cs are pairwise disjoint.

meta_statement('owl':'disjointUnionOf'(C,Cs) ← true).
// Class C is the disjoint union of classes Cs.
```

New meta-statements for expressing new constructs for properties:

```
meta_statement('owl':'propertyDisjointWith'(P,DP) ← true).
// Properties P and DP are disjoint.

meta_statement('owl':'AllDisjointProperties'(Ps) ← true).
// Properties in Ps are pairwise disjoint.

meta_statement('owl':'propertyChainOf'(P,[P1,P2]) ← true).
// Property P is composition of properties P1 and P2.

meta_statement('owl':'objectHasSelf'(C, Pc) ← true).
//Express the Self concept: C is a class of individuals which are related to themselves under role Pc

meta_statement('owl':'reflexive'(P) ← true).
// Property P is reflexive.

meta_statement('owl':'irreflexive'(P) ← true).
// Property P is irreflexive.

meta_statement('owl':'asymmetric'(P) ← true).
// Property P is asymmetric.
```

Figure 6.2 Meta-statements for new OWL 2 features in MMP meta-program

With such new meta-statements together with meta-statements given in chapter 4, DL rules, which are expressed by OWL 2 as we showed in section 3.2.3, can be translated into a meta-program **MMP**. Figure 6.3 show examples of **MMP** that correspond to the DL rules listed in section 3.2.3:

*Rule "$C \sqcap \neg D \sqsubseteq E \sqcup F$" is transformed into **MMP**:*

```
meta_statement('rdfs':'subClassOf'(M,N) ←true).
meta_statement('rdfs':'unionOf'(N,[E,F]) ←true).
meta_statement('rdfs':'intersectionOf'(M,[C,D']) ←true).
meta_statement('rdfs':'complementOf'(D',D) ←true).
```

*Rule "$hasParent \circ hasBrother \sqsubseteq hasUncle$" is transformed into **MMP**:*

```
meta_statement('owl':'propertyChainOf'('f':'hasUncle',
['f':'hasParent','f':'hasBrother']) ←true).
```

*Rule "$Man(x) \wedge hasChild(x,y) \rightarrow fatherOf(x,y)$" is transformed into **MMP**:*

```
meta_statement('owl':'objectHasSelf'
('f':'Man',P_Man) ←true).
meta_statement('owl':'propertyChainOf'
('f':'fatherOf',[P_Man,'f':'hasChild']) ←true).
```

Figure 6.3 Examples of rules in OWL 2 expressed in **MMP**

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

In chapter 4, we gave many axioms in **AMP** to support reasoning about an OWL ontology, and in chapter 5 we have defined axioms in **AMP** to support reasoning about an SWRL ontology. In order to work with rules and ontologies expressed in OWL 2, we *add more axioms* for the new features in OWL 2, axioms for set constructors, and an axiom for negative property. Here we list all of new axioms in **AMP** in Figure 6.4 as follows.

New axioms for new constructs and characteristics of property:

```
axiom('owl':'propertyDisjointWith'(P,DP) ←                    (asdp)
          'owl':'propertyDisjointWith'(DP,P)).
// property disjoint is symmetric.

axiom(P(S,O) ← 'owl':'propertyDisjointWith'(P,DP) ∧           (acdp)
              DP(S,O1) ∧ 'owl':'differentFrom'(O,O1)).
// Properties P and DP are disjoint.

axiom(P(S,O) ← 'owl':'asymmetric'(P) ∧ P(O,S1) ∧              (acasp)
     'owl':'differentFrom'(S,S1)).
// Property P is asymmetric.
```

```
axiom(P(S,S) ← 'owl':'reflexive'(P)).                                    (acrep)
```

// Property P is reflexive.

```
axiom(P(S,O) ← 'owl':'irreflexive'(P) ∧                                  (acirp)
                'owl':'differentFrom'(S,S1)).
```

// Property P is irreflexive.

```
axiom(P(S,O) ← 'owl':'propertyChainOf'(P,[P1,P2]) ∧                      (acpc)
    P1(S,O1) ∧P2(O1,O)).
```

// Axiom to handle property chain: Property P is composition of property P1 and property P2

```
axiom(P(S,S) ← 'owl':'objectHasSelf'(C,P) ∧                              (acsc)
               'rdf':'type'(S,C)).
```

// Axiom to handle the Self Concept.

```
axiom('¬'P(S,O) ← 'owl':'negativeProperty'(P,S,O)).                      (anap)
```

// Axiom to handle a negative property.

New axioms for set constructs:

```
axiom('rdf':'type'(I,C) ←                                                (asic)
          'owl':'intersectionOf'(C,Cs) ∧ 'interType'(I,Cs)).

'interType'(I,[H|T]) ← 'rdf':'type'(I,H) ∧ 'interType'(I,T).
```

// Axiom to handle a set intersection: C is an intersection of all classes in list Cs, I is an instance of C if I instance of each class in Cs.

```
axiom('rdf':'type'(I,C) ←                                                (asuc)
          'owl':'unionOf'(C,Cs) ∧ 'unionType'(I,Cs)).

'unionType'(I,[H|T]) ← 'rdf':'type'(I,H).

'unionType'(I,[H|T]) ← 'unionType'(I,T).
```

// Axiom to handle a set union: C is union of classes in Cs, I is an instance of C if I is instance of at least one class in Cs.

```
axiom('¬' 'rdf':'type'(I,C) ←                                            (ascc)
          'owl':'complementOf'(C, Cc) ∧ 'rdf':'type'(I,Cc)).
```

// Axiom to handle a set complement: If Cc is complement class of C, I is an instance of Cc then I is not instance of C.

Figure 6.4 New axioms in AMP to support OWL 2

6.2 Meta-Interpreter for Reasoning with Meta-programs

Similarity, we adapt the Vanilla meta-interpreter in order to reason with meta-programs transformed from ontologies with rules expressed in OWL 2, where we have defied three kinds of meta-level statements: `statement(A←B)` for the object-level information of an OWL 2 ontology, `meta_statement(A←B)` for the meta-level information of an OWL 2 ontology, and `axiom(A←B)` for supporting axioms to reason with an OWL 2 ontology. Our SW meta-interpreter is defined as follows:

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

The first clause (true) is the basic for proving that an atom is `true`. The second clause (conj) is used for proving a conjunction goal. Three clauses (ost), (mst), and (ast) are used for proving three meta statements of the three meta-programs **MP**, **MMP**, and **AMP** respectively.

6.3 Query Answering OWL 2 ontology with Our Framework

Ontologies and rules expressed in OWL 2 can be reasoned in our framework by firstly OWL 2 ontology be transformed to meta-programs (**MP** and **MMP**), and then our meta-interpreter will manipulate with those meta-programs to infer implicit information.

The family ontology from [41] is used as an example to demonstrate the reasoning ability of our framework. Due to its lack of rules in the ontology, three OWL2 rules are added to it. After the whole ontology is transformed into meta-programs, here are some parts of them:

- The **MP** program

```
statement('f':'hasParent'
          ('f':'M02','f':'M01') ← true).                       (b1)
statement('f':'hasParent'
          ('f':'F02','f':'M01') ← true).                       (b2)
statement('f':'hasParent'
          ('f':'M03','f':'F02') ← true).                       (b3)
```

- **The MMP program**

```
meta_statement('rdf':'type'                                          (b1')
          ('f':'M01','f':'Man')←true).

meta_statement('rdf':'type'                                          (b2')
          ('f':'M02','f':'Man')←true).

meta_statement('rdf':'type'                                          (b3')
          ('f':'F02','f':'WoMan')←true).

meta_statement('owl':'complementOf'                                  (b4')
          ('f':'Man','f':'WoMan')←true).

meta_statement('owl':'unionOf'                                       (b5')
          ('f':'Human',['f':'Man','f':'WoMan'])←true).
```

The first added rule `hasParent(x,y) ∧ hasParent(z,y) → siblingOf(x,z)` is expressed by `hasParent ∘ hasParent⁻ ⊑ siblingOf` DL axiom, where `hasParent⁻` is inverse property of `hasParent`. We transform this into the following meta-statements:

```
meta_statement('owl':'inverseOf'                                     (b6')
          ('f':'parentOf','f':'hasParent')←true).

meta_statement('owl':'propertyChainOf'                               (b7')
   ('f':'siblingOf',['f':'hasParent','f':'parentOf'])←true ).
```

The second rule `Man(x) ∧ siblingOf(x,y) → brotherOf(x,y)` is transformed into the meta-statements in **MMP**:

```
meta_statement('owl':'objectHasSelf'                                 (b8')
          ('f': 'Man', P_Man)←true).

meta_statement('owl':'propertyChainOf'                               (b9')
       ('f':'brotherOf',[P_Man,'f':'siblingOf'])←true).
```

The third rule `brotherOf(x,y) ∧ parentOf(y,z) → uncleOf(x,z)` is transformed into the following meta-statement in **MMP**:

```
meta_statement('owl':'propertyChainOf'                               (b10')
   ('f':'uncleOf',['f':'brotherOf','f':'parentOf'])←true ).
```

Now we then pose some queries to the meta-interpreter and get the answers as follows:

```
?-demo('rdf':'type'('f':'M02',X)).                                   (q1)
  X = 'f':'Man';
  X = 'f':'Human'.
```

//1st answer is supported by (mst), (b1'), and (true), and 2nd answer is supported by (ast), (asuc), (conj), (b5'), (b1'), and (true).

```
?- demo('¬''rdf':'type'('f':'F02',X)).                               (q2)
  X = 'f':'Man'.
```

//The adopted clauses are (ast), (ascc), (conj), (mst), (b4'), (b3') and (true).

```
?- demo('f':'siblingOf'('f':'M02',X)).
  X = 'f':'F02'.
```
(q3)

//The adopted clauses are (ast), (acpc), (conj), (b7'), (true), (ost), (b1), (mst), (acip), (b6'), (b2).

```
?- demo('f':'brotherOf'(X, 'f':'F02')).
  X = 'f':'M02'.
```
(q4)

//The adopted clauses are (ast), (acpc), (conj), (b9'), (true), (acsc), (b8'), (mst), (b2'), and the clauses adopted for answering q3.

```
?- demo('f':'uncleOf'('f':'M02',X)).
  X = 'f':'M03'.
```
(q5)

//The adopted clauses are (ast), (acpc), (conj), (b10'), (true), (b3), and the clauses adopted for answering q4.

6.4 Conclusion

We have presented a meta-logical framework for representing and reasoning with ontologies and rules expressed in OWL 2. The logical system of our framework consists of meta-programs transformed from ontologies and rules expressed in OWL 2, and an inference engine defined by a demo predicate with the new extra auxiliary axioms proposed in the chapter.

Chapter 7

A Unified Meta-logical Framework for Reasoning with Semantic Web Ontologies with Rules

7.1 Introduction

In the previous chapters meta-logical frameworks for manipulating SW ontologies with rules expressed in SWRL and OWL 2 are described. In this chapter we unify them to define one single framework which can reason with Semantic Web Ontologies with Rules.

7.2 Abstract Syntax Representation of MP/MMP Meta-statements

SW ontologies with rules are transformed into MP and MMP meta-statements which have abstract syntax representation in the terms of meta-predicate `statement(S)` and `meta_statement(S)`.

7.2.1 Abstract syntax representation of MP meta-statements

As we described in chapter 5, and chapter 6 syntax for describing meta-statements of object-level information of an OWL 2 ontology is in the form of `statement(P(S,O) ← true)`, while for describing meta-statements of object-level information of an SWRL ontology, the form `statement(P(S,O) ← true)` is used describe object-level information of the ontology, for a SWRL rule we use form `statement(P(S,O) ← Body)` to expresses this rule. Put both of the forms together we have abstract syntax representation of MP meta-statements as follows

```
statement(P(S,O) ← true).
statement(P(S,O) ← Body).
```

7.2.2 Abstract syntax representation of MMP meta-statements

In chapter 6 we defined there forms of MMP meta-statements `meta_statement(P(S,O) ← true)`, `meta_statement(P(S,Os) ← true)`, and `meta_statement(C(Prop) ← true)` in order to describe meta-level information of an OWL 2 ontology. With these there forms, rules expressed in OWL 2

can also be expressed in MMP. However, for a SWRL rule describing a meta-information such as relationship between an individual and a class we defined one more form of MMP meta-statement `meta_statement(P(S,O) ← Body)`. So in order to describe meta-information of SW ontologies with rules we have abstract syntax representation of MMP meta-statements as follows.

```
meta_statement(P(S,O) ← true).
meta_statement(P(S,O) ← Body).
meta_statement(P(S,Os) ← true).
meta_statement(C(Prop) ← true).
```

7.2.3 Translating SW Ontology with Rules into Meta-programs

Using the abstract syntax of meta-languages above, an SW ontology with rules expressed in either SWRL or OWL 2 can be translated into meta-level statements in **MP** and **MMP**. A rule expressed in SWRL is translated into a meta-statement `statement(A ← B)` in **MP** if it is a rule of object information of an SW ontology, or into a meta-statement `meta_statement(A ← B)` in **MMP** if it is a rule of meta information of an SW ontology. A rule expressed in OWL 2 is translated into meta-statements in **MMP**.

7.3 Meta-program for Axioms (AMP)

A meta-program **AMP**, which is defined for mathematic axioms, is needed for reasoning process of the meta-interpreter.

For SWRL ontology, in order to process built-in functions we defined a form as follows.

`axiom(builtinAtom(Fc,Arg1,...Argn) ← builtin(FcinPrlog)).`

This axiom supports a mapping between a SWRL built-function and a corresponding Prolog predicate.

For OWL 2 ontology we have defined axioms to cope with new features which we can use to express rules. Figure 6.4 shows all of new axioms in AMP to support OWL 2.

Put all of these axioms together, we have a complete AMP program to support the meta-interpreter for reasoning with SW ontologies with rules.

7.4 Meta-interpreter

The definition of meta-interpreter in chapter 5 for reasoning with SWRL ontologies, and the definition of meta-interpreter in chapter 6 for reasoning with OWL 2 ontologies are almost the same. The meta-interpreter for reasoning with SWRL ontologies cqvers all the definition of the meta-interpreter for reasoning with OWL 2 ontologies, however it has an extra clause (btin) which was defined to manipulate with SWRL built-in functions. Therefore, by using the meta-interpreter defined in chapter 5, our meta-framework has full ability for reasoning with SW ontologies with rules expressed in either SWRL or OWL 2.

7.5 Conclusion

We have briefly presented the general framework for reasoning with SW ontologies with rules in this chapter. With capability of reasoning with SW ontologies and rules expressed in SWRL and OWL 2, we have extended the MAC-SWI framework in order to work with more expressive SW languages for describing semantic web ontology with rules.

Chapter 8

Comparison with Related Works

8.1 Introduction

In the previous chapters we have described in details our meta-logical framework for reasoning with SW ontologies with rules. In this chapter we will compare our approach with other related works done by other. The comparison will focus on important issues such as formal representation of SW ontologies with rules, and the ability to reason with SW ontologies.

8.2 The Proposal and Other Logic Programming Approaches

The SW research community has addressed similar issues and problems concerning SW ontologies and rules as that also happen in the area of logic programming. So the exchange of idea between these two research areas is inevitable.

- **Description Logic Programs: Combining Logic Programs with Description Logic**

Realizing the necessity of integration between rules (RuleML) and SW ontologies (OWL/DAML+OIL), Grosof, and Horrocks [18] proposed a form of knowledge representation, called 'Description Logic Program' (DLP), which employs rules and OWL DL (OWL Description Logic) ontologies. Their approach supports bi-directional translation between logical sentences from DLP fragment of Description Logic and logic programs (in logic programming).

According to this approach, every concept referred to in an ontology is mapped into a unary relation with the concept name becoming the name of the relation and the individual name becoming the argument. Every instance-property-instance relationship is mapped into a binary relation. In addition, concepts as well as property constructor statements are converted into rules. The distinctions between this approach and ours are the following.

Firstly this approach was designed to support a subset of DAML+OIL, and provides only a mapping from RDFS and DAML+OIL to logic programs, but does not support the new features in OWL 2, as well as SWRL ontology. Secondly, this approach has a weakness to represent an ontology in a logic program. For example, to represent the statement "a is union of b_1, b_2, ..., b_n", it requires n rules to do so, i.e. a(X):-b_1(X), ..., a(X):-b_n(X). However, in our representation, this requires only one statement: `meta_statement('owl':'unionOf'(a, [b1,...,bn]) ← true)` which is more compact.

Even more importantly, with their representation of logic program, concept names and property names of an ontology cannot be accessed from the logic program. For example, two statements saying that "John is an instance of class Man" and "John is son of Mary" are mapped into the following facts:

```
Man('John').
hasSon('Mary', 'John').
```

With these facts we can ask who is a man or who is a son of Mary, but it is impossible to get answers to a question like which class that John is an instance of or what a relationship between Mary and John is. This is because a predicate name is a meta-level information that cannot be reasoned and queried at object-level by a Prolog interpreter. However, in our approach since we separate the meta-level from the object-level knowledge in an ontology, above examples are expressed by two meta-statements as follows:

```
meta_statement('rdf':'type'('John', 'Man') ← true).
statement('hasSon'('Mary', 'John') ← true).
```

Such queries above can be easily asked and answered via our meta-interpreter because it can access and manipulate the names of those classes which become arguments of predicates in our meta-representation:

```
?-demo('rdf':'type'('John', X)).
    X= 'Man'.

?-demo(P('Mary', 'John')).
    P= 'hasSon'.
```

- **SweetProlog: A System for translating an OWL ontology and rules into a Prolog program**

Laera et al. [27] proposed SweetProlog as a system for translating an OWL ontology and rules into a Prolog program. It is achieved by the translation of an OWL ontology described in Description Logic and rules expressed in OWLRuleML into a set of facts and a set of rules in Prolog respectively. Then any reasoning on these facts and rules can be performed by a Prolog interpreter.

SweetProlog is implemented in Java and makes use of the three languages: Prolog as rule engine, OWL as ontology language and OWLRuleML as a rule language that reasons over the data model of OWL, and a set of inference rules that translated OWL into Prolog rules. It enables reasoning over OWL ontologies by rule via a translation of OWL subset into simple Prolog predicates. SweetProlog consists of five principal functions as follows:

- Translation of the OWL and OWLRuleML ontologies into RDF triples: SweetProlog reads an OWL ontology and OWLRuleML rules, and extracts RDF triples out of the ontology.
- Translation of the OWL assertion into Prolog clauses: The extracted RDF triples that represent OWL concepts and instances are translated into Prolog predicates.
- Translation OWLRuleML rules into Prolog rules: The RDF triples that present the rules are translated into Prolog rules.
- Reasoning with Prolog program: Finally, both the Prolog predicates and Prolog rules form a Prolog program that can be reasoned by a Prolog interpreter.

This approach develops an idea to specify RuleML on top of OWL ontologies. It provides semantic and inferential interoperation between ontologies and rules. An immediate result of this research is that it is possible to reason over the instances of an ontology with rule definition that uses vocabulary from the ontology by using a Prolog interpreter.

Comparing this with our work, according to their approach an OWL and RuleML ontology is entirely translated into Prolog facts and rules, where both object level and meta level knowledge of the ontology are mixed up; Laera et al. do not care to make the distinction between the two levels of knowledge, whilst our translation makes a careful separation between the two levels of knowledge. As a result, their SweetProlog can reason with any object level

knowledge of an ontology as the way our approach does.

For example, with the same ontology that we use for the query answering in Section 4, according to their approach, the ontology would be transformed to Prolog facts as follows:

```
hasParent('f':'F02','f': 'M01').
hasBrother('f': 'M01', 'f': 'M03').
...
```

and the 'uncle' rule would be expressed by the Prolog rule:

```
hasUncle(X,Y):- hasParent(X,Z), hasBrother(Z,Y).
```

Provided with this prolog program and a query like

```
?-hasUncle('f':'F02', X),
```

their SweetProlog would give the answer `X='f':'M03'`, which is the same answer as that given earlier by our meta-interpreter.

However, there could be queries, which ask about meta-level information of this ontology, which their SweetProlog cannot give answers to, since there is some information that can be asked only at the meta level, but cannot do that at the object level.

For example, a query asking what the relation between `M02` and `M03` is:

```
?-P('f':'F02','f':'M03')
```

So, in SweetProlog, the Prolog interpreter will signal *a syntax error*, since a variable is not allowed to be used as a predicate name in a query. Here a predicate name is a meta level information that cannot be asked at the object level.

However, with a careful treatment of a separation between the object level and the meta-level knowledge in our approach, such a query can be asked via the `demo` predicate as follows.

```
?-demo(P('f':'M02','f':'M03')).
```

and our meta-interpreter can give the answer: `P = hasUcle.`

- **SWORIER: Semantic Web Ontologies and Rules for Interoperability with Efficient Reasoning**

Samuel et al. [35] proposed SWORIER (Semantic Web Ontologies and Rules for Interoperability with Efficient Reasoning) system, which enables efficient automated reasoning on ontologies and rules, by translating all of ontologies and rules into a Prolog program and adding a set of general rules that formulate the OWL primitives. A Prolog interpreter is used to reason with the resulting Prolog program to derive answers.

Figure 8.1 shows the system design of SWORIER, the ontologies and rules are created by developer by using OWL and SWRL. These ontologies and rules together are translated into Prolog codes using XSLTs (Extensible Stylesheet Language Transformations). Finally, a set of *General rules* is appended to the XSLT output to form a complete Prolog program which can be queried by the user using a Prolog interpreter.

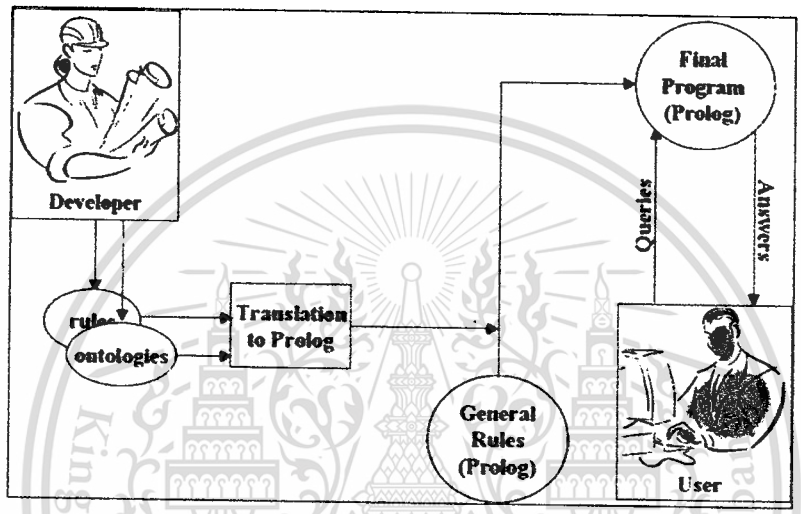


Figure 8.1 Design of SWORIER

In SWORIER, two Prolog predicates `ismemberof/2` and `haspropertywith/3` are defined in order to translate ABox knowledge of ontology into Prolog facts, i.e. a class assertion *C(a)* will be translated into a Prolog fact `ismemberof(a,C)` and a instance-property-instance relationship *P(a,b)* will be translated into a Prolog fact `haspropertywith(a,P,b)`. By using these two predicates some queries, which cannot answer by SweetProlog, such as queries asking about the class which an instance belongs to or the relationship between two individuals now can be asked and answered by SWORIER.

In addition, in SWORIER Samuel et al. defined a set of *General Rules* in Prolog in order to formulate the OWL primitives. Here their *General Rules* serve the same purpose as our **AMP** program, however our **AMP** also supports many axioms for manipulating with SWRL built-in functions as well as the new features in OWL 2, which are not supported in their framework.

8.3 Conclusion

In this chapter we have done a comparison between our framework with other approaches that related to logic programming. The results of the comparison reveals an advanced reasoning ability of our framework over those approaches; for example the separation between object-level and meta-level of SW ontologies, and supporting SW languages for describing SW ontologies with rules.

Chapter 9

Conclusion

This thesis proposes a meta-logical framework for reasoning with Semantic Web ontologies with rules expressed in SWRL and OWL 2. The framework is an enhancement of the MAC-SWI framework which was designed for reasoning with OWL ontologies. In this chapter, we conclude main results achieved in the thesis, and we then give an outlook on future work of this research.

9.1 Main Results

The main results of the research work are summarized as follows.

- **Meta-languages for expressing SW ontologies with rules**

We first we enhance meta-languages in the MAC-SWI, which include two meta-languages (**ML** and **MML**) for expressing two levels of information of ontologies (object-level information and meta-level information respectively) in terms of meta-logical statements. With our enhancement the meta-languages support syntaxes for expressing Horn-clause rules, SWRL built-in functions in SWRL ontologies as well as the new features in OWL 2 such as negative property assertion, property chain. This enhancement allows the representation of SW ontologies and rules in a uniform manner which enables the development of efficient and effective reasoning systems.

- **Meta-programs for representing SW ontologies with rules**

Meta-programs are knowledge base of the inference engine in our framework. They contain meta-statements which are transformed from SW ontologies with rules using the meta-languages (**ML** and **MML**). Regarding to the two information levels of ontologies, there are two meta-programs **MP** and **MMP**. Horn-clause rules in SWRL ontologies can be described in both **MP** and **MMP**, since these rules can be either object-level relationships or meta-level relationships. For rules expressed in OWL 2 are

transformed into meta-statements in **MMP**. To be able to reason with these meta-programs, the inference engine often requires another meta-program the **AMP** meta-program for mathematical axioms. These three meta-programs serve as input for the meta-interpreter which reasons and infers conclusions from the meta-programs.

- **Auxiliary axioms for SWRL built-ins and new features in OWL 2**

In order for our framework to work with SWRL and OWL 2, we also defined more auxiliary axioms in **AMP** to formulate SWRL built-ins and new features in OWL 2. Axioms for SWRL built-ins support the correspondence between built-in functions with Prolog built-in predicates, so that our meta-interpreter can easily reason with SWRL built-ins. In addition, more axioms for capturing semantic of new features in OWL 2, such as negative property, disjoint properties, property chain, etc., are also defined in **AMP**. With the supports of auxiliary axioms, the meta-interpreter can reason with SW ontologies and rules expressed in either SWRL or OWL 2.

- **Meta-interpreter for reasoning with SW ontologies with rule expressed in SWRL and OWL 2**

We have slightly adapted and extended the well-known meta-interpreter in logic programming—The Vanilla interpreter—to reason with SW ontologies with rules. This extension of the meta-interpreter consists of 1-ary demo predicate with definition of six clauses which are able to not just include inference rules for reasoning with various meta-statements of different meta-programs transformed from SW ontologies with rules, but also contain rule for exchanging the SWRL built-in functions to the corresponding built-in Prolog predicates. With these abilities, our meta-interpreter can be considered as an inference engine in our framework for reasoning with SW ontologies with rules.

9.2 Future Works

This thesis focuses on development of a meta-logical framework for reasoning with SW ontologies with rules expressed in SWRL and OWL 2. However, recently both SWRL and OWL 2 are not perfects way to express SW ontologies with rules. As we mentioned in chapter

3, the direct addition of the rules to ontologies causes the undecidability when reasoning with SWRL ontologies. Rules in OWL 2 are a decidable fragment of SWRL by putting upon some restrictions to SWRL. These restrictions aim to restrict a logical rule to be a DL rule, that is, rule used in Description Logic context. Contrary to the field of logic programing, such kind of rules does not have widely acknowledged syntax and semantic, so there are rules which can easily expressed in logic programming but cannot be expressed in OWL 2. It is immediately clear that a single language would not satisfy the needs of many popular paradigms of using rules in knowledge representation and business modeling. Known rule systems fall into three broad categories: DL rules, logic programming, and action rules. These paradigms share a little in the way of syntax and semantic. Moreover, there are large differences between systems even within the same paradigm. This diversity of rule systems leads to a notion of rule exchange. The approach recently taken by W3C Working Group was to design a family of languages, called Rule Interchange Format dialects [45]. The central idea behind rule exchange through RIF is that different systems will provide syntax mappings from their native languages to RIF and vice versa. These mappings are required to be semantics-preserving, and thus rule sets can be communicated from one system to another provided that the systems can talk through a suitable dialect, which they both support. With the development of RIF, in order to support our framework to communicate with other rule systems, an extension of our framework to work with RIF would be one of our future works.

As we said, SWRL is undecidable, that is there is no algorithm that can, in finite time, compute the whether an axiom is entailed by a SWRL knowledge base. It is, in fact, semi-decidable—if an axiom entailed, then there is an algorithm that can draw conclusion from the axiom; but if an axiom is not entailed, it's possible to "go into an infinite loop", i.e., not to terminate. DL safety is a simple idea has been used to regain decidability: *Variable in DL safe rules bind only to explicit named individuals in ontology*. Adding this restriction make SWRL make SWRL rules decidable. It shifts SWRL rules from being a kind of supper powerful TBox and RBox (i.e., class and property) axiom to them being a sort of data manipulation (i.e., ABox) axiom. Therefore, variety of knowledge would be hard to express into ontologies and rules by using the restriction. With that problem, we figure out that a significant improvement in our future works is find a way to restrict SWRL rules at syntactic level so that it have more freedom to describe knowledge compare with the previous restriction. The restricted SWRL rules, then,

can be expressed as meta-programs in our framework, and these meta-programs will be proved that they are soundly and completely reasoned by our meta-interpreter.

Another area for future extension is the incorporation of fuzziness. The need to deal with vague information in Semantic Web languages is rising in importance, thus, calls for a standard way to represent such information. An OWL ontology to present fuzzy extensions of OWL 2 language has been proposed in [42]. And a fuzzy extension to SWRL to include fuzzy assertions and fuzzy rules is proposed in [43, 44]. The widely use of such type of imprecise information in many applications like multimedia processing is an additional motivation for pursing further research to extend our framework in this direction.

REFERENCES

[1] V. Hirankitti and V. T. Xuan, "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*, Ho Chi Minh City, Vietnam February, 2006, pp. 228-235.

[2] V. Hirankitti and V. T. Xuan, "Meta-reasoning with Multiple Distributed Ontologies on the Semanttic Web," in *proc. of the 8th International Conference on Intelligent Technologies*, Assumption University, Phuket, Thailand, December, 2005, pp. 301-309.

[3] V. Hirankitti and V. T. Xuan, "Semantic Web Agent Communication Capable of Reasonig with Ontology and Agent Locations," in *proc. of the 8th International Conference on Cybernetics, Informatics, and Systemics*, Krakow, Poland, December 2005, pp. 98-104.

[4] V. Hirankitti and V. T. Xuan, "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*, Waseda University, Fukuoka, Japan, October, 2005, pp. 7-16.

[5] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel Schneider, *The Description Logic Hand book: Theory, Implementation, and Applications*, 2 ed.: Cambridge, 2007.

[6] S. Bechhofer, F. v. Harmelen, J. Hendler, I. horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein, "OWL Web Ontology Language Reference," URL http://www.w3.org/TR/owl-ref/, Feb 2004.

[7] D. Beckett, "RDF/XML Syntax Specification," W3C Recommendation, URL http://www.w3.org/TR/rdf-syntax-grammar, 10 February 2004.

[8] T. Berners-Lee, "Notation 3 (N3) A readable RDF Syntax," URL http://www.w3.org/DesignIssues/Notation3.html, 1998.

[9] T. Berners-Lee, "Realising the Full Potential of the Web," W3C Document, Dec 1997.

[10] T. Berners-Lee, "Semantic Web Road Map," W3C Design Issues, Oct. 1998.

[11] T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web," *Scientific American*, pp. 34-43, 2001.

[12] T. Bray, D. Hollander, A. Layman, and R. Tobin, "Namespaces in XML 1.0 (second edition)," W3C Recommendation, 16 August 2006.

[13] D. Brickley and R. V. Guha, "RDF Vocabulary Description Language 1.0:RDF Schema," URL http://www.w3.org/TR/rdf-schema/, 2004.

[14] J. Cowan and R. Tobin, "XML Information Set (second edition)," Technical report, W3C, 2004.

[15] F. M. Donini, M. Lenzerini, and A. Schaerf, "AL-log: Integreating datalog and description logics," *Journal of Interlligent Informatin Systems,* pp. 227-252, 1998.

[16] T. Eiter, T. Lukasiewicz, R. Schindlauer, and H. Tompits, "Combining Answer Set Programming with Description Logics for the Semantic Web," in *Proc. Ninth International Conference of Pronciples of Knowledge Representation and Reasoning (KR2004)*, 2004, pp. 141-151.

[17] F. Gasse, U. Sattler, and V. Hachinski, "Rewriting Rules into SROIQ Axioms," in *Poster at 21st Int. Workshop on DLs (DL-08)*, 2008.

[18] B. N. Grosof, I. Horrocks, R. Volz, and S. Decker, "Description Logic Programs: Combining Logic Programs with Description Logic," *In Proc. of the 12th Int'l Conf. on the WWW, ACM,* pp. 48-57, 2003.

[19] J. W. C. I. U. P. I. Group, "URIs, URLs, and URNs: Clarifications and Recommendations 1.0," URL http://www.w3.org/TR/uri-clarification, W3C Note, 2001.

[20] T. R. Gruber, "Towards Princeples for the Design of Ontologies Used for Knowledge Sharing," In N, Guarino and R. Poli, editeors, *Formal Ontology in Conceptual Analysis and Knowledge Representation*, Deventer, The Netherlands, 1993.

[21] M. Hori, J. Euzenat, and P. F. Patel-Schneider, "OWL web ontology language XML presentation syntax," W3C Note, 11 June 2003. URL http://www.w3.org/TR/owl-xmlsyntax.

[22] I. Horrocks, O. Kutz, and U. Sattler, "The even more irressistible SROIQ," in *Proc. of the 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR2006)*: AAAI Press, 2006, pp. 57-67.

[23] I. Horrocks and P. F. Patel-Schneider, "A Proposal for an OWL Rules Language," in *Proc. of the 13th Int'l Conf. on the WWW, ACM*, 2004.

[24] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. N. Grosof, and M. Dean, "SWRL: A semantic web rule language combining owl and ruleml," W3C Note, 21 May 2004. URL http://www.w3.org/Submission/SWRL/.

[25] I. Horrocks, P. F. Patel-Schneider, and F. v. Harmelen, "Reviewing the Design of DAML+OIL: An Ontology Language for the Semantic Web," in *Proc. of the 18th Nat. Conf. on Artificial Intelligence (AAAI 2002)*, 2002.

[26] R. A. Kowalski and J. S. Kim, "A Metalogic Programming Approach to Multi-agent Knowledge and Belief," *In AI and Mathematical Theory of Computation,* pp. 231-346, 1991.

[27] L. Laera, V. Tamma, T. Bench-Capon, and G. Semeraro, "SweetProlog: A System to Integrate Ontologies and Rules," in *Proc. Of RuleML'04*, 2004, pp. 188-193, Springer Verlag.

[28] A. Y. Levy and M.-C. Rousset, "Combining Horn rules and description logics in CARIN," *Artificial Intelligence,* pp. 165-209, 1998.

[29] J. Marsh and R. Tobin, "XML base (second edition)," W3C Recommendation, URL http://www.w3.org/TR/2008/PER-xmlbase-20080320/, March 2008.

[30] B. Motik, U. Sattler, and R. Studer, "Query answering for OWL-DL with rules," *Journal of Web Semantics: Science, Services and Agents on the World Wide Web,* pp. 41-60, 2005.

[31] B. Nebel, "Terminological Reasoning is Inherently Intractable," *Artificial Intelligence,* pp. 235-249, 1990.

[32] B. Nebel, "Terminological Cycles: Semantics and Computational Properties," in *J. F. Sowa, Principles of Semantic Networks: Exploration in the Representation of Knowledge*: Morgan Kaufmann Publishers, San Mateo (CA), USA, 1991, pp. 331-361.

[33] P. F. Patel-Schneider, P. Hayes, and I. Horrocks, "OWL Web Ontology Language Semantics and Abstract Syntax," W3C Recommendation, 10 February 2004.

[34] P. F. Patel-Schneider and B. Motik, "OWL 2 Web Ontology Language: Document Overview," W3C Recommendation, 27 October 2009.

[35] K. Samuel, L. Obrst, S. Stoutenberg, K. Fox, P. Franklin, A. Johnson, K. Laskey, D. Nichols, S. Lopez, and J. Peterson, "Translating OWL and Semantic Web Rules into

Prolog: Moving toward description logic programs," *Jounal Theory and Practice of Logic Programming,* pp. 301-322, 2008.

[36] M. Schmidt-Schauß and G. Smolka, "Attributive concept descriptions with complements," *Artificial Intelligence,* vol. 48, pp. 1-26, 1991.

[37] V. T. Xuan, "A Meta-logical Framework for Agent Communication of Semantic Web Information," Mater thesis in School of Graduate Studies, King Mongkut's Institute of Technology Ladkrabang, Bangkok, Thailand, 2005.

[38] "Rule Markup Language," URL, http://www.ruleml.org/.

[39] B. Motik, P. F. Patel-Schneider, and B. Parsia, "OWL 2 web ontology language document overview," URL, http://www.w3.org/TR/owl2-overview/. 2009.

[40] "The family SWRL ontology," URL, http://protege.cim3.net/file/pub/ontologies/family.swrl.owl/family.swrl.owl/.

[41] "The family ontology for testing new features in OWL 2," URL, http://www.owldl.com/ontologies/family.owl.

[42] F. Bobillo and U. Straccia, "An OWL Ontology for Fuzzy OWL 2," in *ISMIS '09 Proceedings of the 18th International Symposium on Foundations of Intelligent Systems*, 2009, pp. 151-160.

[43] X. Wang, X. M. Ma, L. Yan, and X. Meng, "Vague-SWRL: A Fuzzy Extension of SWRL," in *RR '08 Proceedings of the 2nd International Conference on Web Reasoning and Rule Systems*, 2008, pp. 232-233.

[44] J. Z. Pan, G. Stamou, V. Tzouvaras, and i. Horrocks, "f-SWRL: A Fuzzy Extension of SWRL," *Journal of Data Semantic, special issue on Emergent Semantic,* pp. 28-46, 2006.

[45] M. Kifer and H. Boley, "Rule Interchange Format (RIF): RIF Overview," W3C Working Group, June 22, 2010.

Appendix I Implementation of a Meta-logical Framework for Reasoning with SW Ontologies with Rules

I.1 Introduction

A prototype of our meta-logical framework has been implemented. Our framework is a system for answering information about an SW ontology with rules. In this appendix we will present details the implementation of the system. First we give an introduction of system components used to support the development, deployment, and testing of our system such as operating system, programming languages, software toolkits as well as client software and server software. A short summary of applied functions and the role of these toolkits will be given. We then present the implementation of some main parts such as the ontology, ontology transformation, and the meta-interpreter.

I.2 Development Infrastructure

Important components used for developing, testing, and deploying our system are listed as follows.

- Operating system: Windows XP/ Windows 7, Mac OS.
- Software toolkits: Win Prolog 4.9, SWI-Prolog, Protégé 4.1, etc.
- Programming languages: Prolog, Web programming languages (HTML, JavaScript, etc.).
- Server: IIS web server, LPA Proweb 4.9.
- Client: Internet Explorer, Firefox, etc.

In the following we summarize the main functions of these toolkits.

- **Win-Prolog** plays the important role for both development and deployment of our system. It is a 32-bit Prolog compiler and programming environment conforming to the Edinburgh standard. Codes of our system are written in Prolog in this standard, and are compiled by Win-Prolog. Furthermore Win-prolog provides many tools supporting the development of web-based, network-based, or agent-based applications. They are very useful for the development of our system.

- **SWI-Prolog** is a development environment for developing Prolog applications. SWI-Prolog provides a RDF library for parsing RDF documents. This library is used for developing our ontology transformation module.

- **LPA Proweb server:** Proweb is a specialized extension to a Hypertext Transport Protocol (HTTP) server. Proweb is written in Prolog and supports the development, testing, and deployment of intelligent, dynamic server-based applications on intranets and the internet. ProWeb provides a link between an HTML page displayed on a client's web browser and an application (developed in WIN-Prolog) running on a server. ProWeb brings the power of the WIN-PROLOG to the web. Proweb is the main tool to build our website.

- **IIS Web server:** IIS web server is a web server is used in our system to deploy Proweb Server, so that our website can be hosted on the web server.

- **Protégé:** Protégé is software which is developed by Standford Medical Informatics. It is built by using Java programming. It is used for building and editing ontologies. Our hotel ontology is built by using Protégé 4.1.

I.3 Developing Hotel Ontology

The hotel ontology is used as a foundation for semantic web representation of a wide range of hotel information. It encompasses terms and concepts for expressing contact data (address, phone, etc.), general hotel information (number of rooms, place nearby, etc.), price information w.r.t room categories, as well as hotel ratings. It also contains rules for derive more information, such as with price information and room categories we can infer category of hotel by using rules. Figure I.1 shows a partial diagram of the hotel ontology. This hotel ontology is created and edited by using Protégé software, the interface of Protégé with this hotel ontology is depicted in Figure I.2.

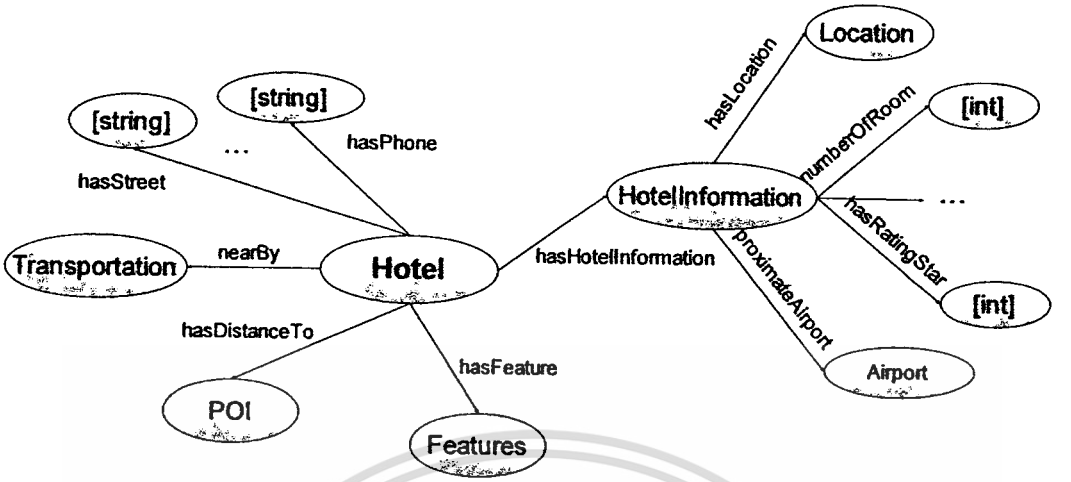


Figure I.1 Partial diagram of hotel ontology

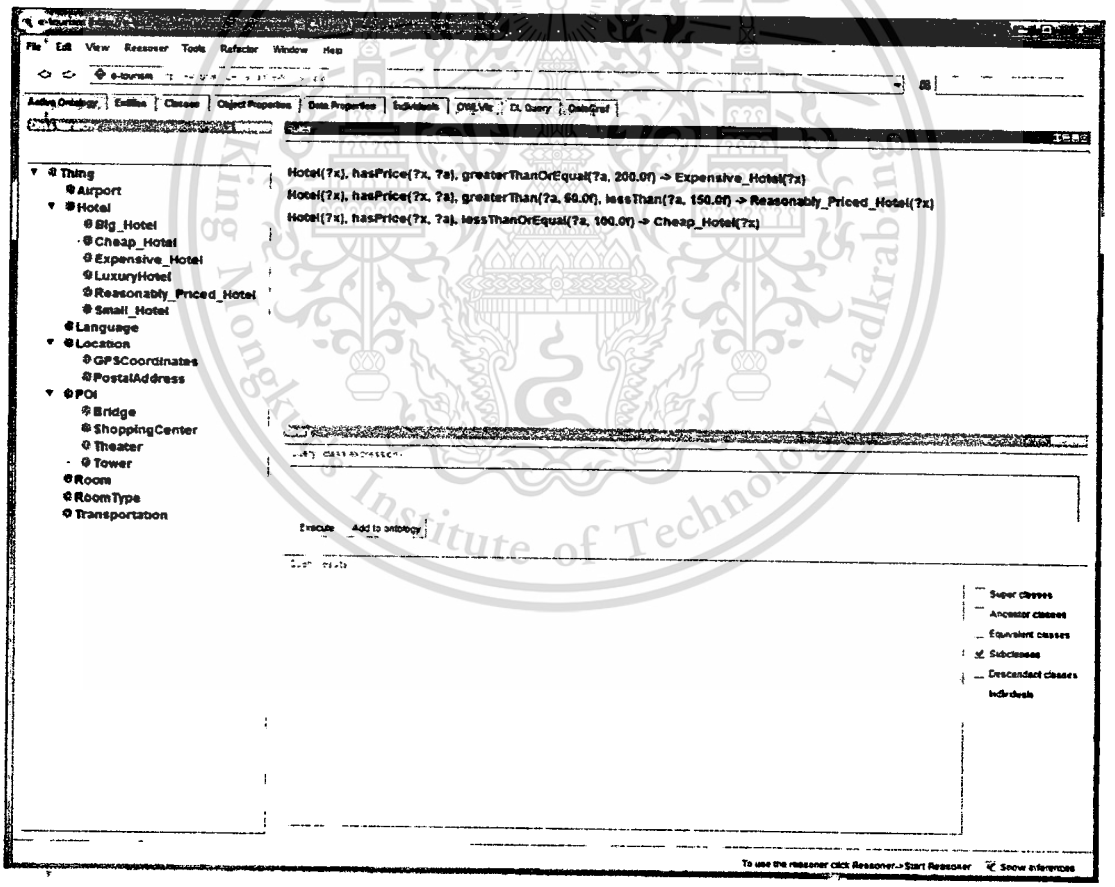


Figure I.2 Protégé Interface with hotel ontology

When searching for a hotel people not only want to know the explicit information about the ontology but also they want to know implicit information which can be inferred by rules. For example, when we have rental price information of the hotels, the system should be able to

infer their categories (e.g. cheap hotels, expensive hotels, reasonably priced hotels). Our system will reason with the ontology with rules in order to infer more information about hotel for user. Figure I.3 shows some examples of rules that we implemented in our system. These rules can be created by using Protégé with Rule taps as shown in Figure I.2.

Term	SWRL Expression
Cheap Hotel	Hotel(?a) ∧ hasRoom(?a, ?b) ∧ roomType(?b, "Double Room") ∧ hasPrice(?b, ?p) ∧ swrlb:lessThanOrEqual(?p, 120) → Cheap_Hotel(?a)
Expensive Hotel	Hotel(?a) ∧ hasRoom(?a, ?b) ∧ roomType(?b, "Double Room") ∧ hasPrice(?b, ?p) ∧ swrlb:greaterThanOrEqual(?p, 200) → Expensive_Hotel(?a)
Reasonably Priced Hotel	Hotel(?a) ∧ hasRoom(?a, ?b) ∧ roomType(?b, "Double Room") ∧ hasPrice(?b, ?p) ∧ swrlb:greaterThanOrEqual(?p, 120) ∧ swrlb:lessThan(?p, 200) → Expensive_Hotel(?a)
Big Hotel	Hotel(?a) ∧ numberOfRoom(?a, ?b) ∧ swrlb:greaterThanOrEqual(?b, 200) → Big_Hotel(?a)
Medium Hotel	Hotel(?a) ∧ numberOfRoom(?a, ?b) ∧ swrlb:greaterThan(?b, 150) ∧ swrlb:lessThan(?b, 200) → Medium_Hotel(?a)
Small Hotel	Hotel(?a) ∧ numberOfRoom(?a, ?b) ∧ swrlb:lessThanOrEqual(?b, 150) → Small_Hotel(?a)

Figure I.3 Rules for a Hotel Ontology

I.4 Ontology Transformation

After we create a hotel ontology by using Protégé, it is in form of OWL. Our transformation module, which was developed using SWI-Prolog, is used to transform this ontology into meta-representation in terms of meta-programs **MP** and **MMP** according to the abstract syntax representation of **MP** and **MMP** meta-statements as we defined in this thesis.

- Abstract syntax representation of MP meta-statements

  `statement(P(S,O) ← true)`, and

  `statement(P(S,O) ← Body)`,

- **Abstract syntax representation of MMP meta-statements**

  `meta_statement(P(S,O) ← true)`,

  `meta_statement(P(S,O) ← Body)`,

  `meta_statement(P(S,Os) ← true)`, and

  `meta_statement(C(Prop) ← true)`,

Figure I.4 shows the **MP** and **MMP** which are transformed from hotel ontology, Figure I.5 shows the rules in Figure I.3 expressed in **MP** and **MMP**. **MP** and **MMP** are expressed in simple logical form so that WIN-PROLOG can manipulate them.

```
% ************************************************************************
%                  Hotel ONTOLOGY
% ************************************************************************

%-------------------------------------------------------------------
% operator declaration

:- op(888, xfy, and).
:- op(999, xfx, if).
:- op(555, yfy, @).
:- op(666, xfx, @).

%-------------------------------------------------------------------
statement(hotelOnto, 'f'@'hasRatingStar'('f'@'Amari Boulevard Bangkok',4) if true).
statement(hotelOnto, 'f'@'numberOfRoom'('f'@'Amari Boulevard Bangkok',309) if true).
statement(hotelOnto, 'f'@'hasRoom'('f'@'Amari Boulevard Bangkok','f'@'ABBGuest1') if true).
statement(hotelOnto, 'f'@'hasRoom'('f'@'Amari Boulevard Bangkok','f'@'ABBGuest2') if true).
statement(hotelOnto, 'f'@'hasLocation'('f'@'Amari Boulevard Bangkok','Bangkok') if true).
statement(hotelOnto, 'f'@'hasAddress'('f'@'Amari Boulevard Bangkok','2 Soi 5, Sukhumvit Road') if true).
statement(hotelOnto, 'f'@'proximateAirport'('f'@'Amari Boulevard Bangkok','f'@'Suwanabumme Airport') if true).

statement(hotelOnto, 'f'@'degreeOfComfort'('f'@'Amari Boulevard Bangkok',7.1) if true).
statement(hotelOnto, 'f'@'degreeOfServices'('f'@'Amari Boulevard Bangkok',7.1) if true).
statement(hotelOnto, 'f'@'degreeOfStaff'('f'@'Amari Boulevard Bangkok',7.7) if true).

statement(hotelOnto, 'f'@'roomType'('f'@'ABBGuest1','Double Room') if true).
statement(hotelOnto, 'f'@'hasPrice'('f'@'ABBGuest1',132) if true).
statement(hotelOnto, 'f'@'hasView'('f'@'ABBGuest1','City') if true).

statement(hotelOnto, 'f'@'roomType'('f'@'ABBGuest2','Deluxe Double') if true).
statement(hotelOnto, 'f'@'hasPrice'('f'@'ABBGuest2',155) if true).
statement(hotelOnto, 'f'@'hasView'('f'@'ABBGuest2','River') if true).

%----------------
statement(hotelOnto, 'f'@'hasRatingStar'('f'@'Column Bangkok',5) if true).
statement(hotelOnto, 'f'@'numberOfRoom'('f'@'Column Bangkok',238) if true).
statement(hotelOnto, 'f'@'hasRoom'('f'@'Column Bangkok','f'@'CBGuest1') if true).
statement(hotelOnto, 'f'@'hasRoom'('f'@'Column Bangkok','f'@'CBGuest2') if true).
statement(hotelOnto, 'f'@'hasLocation'('f'@'Column Bangkok','Bangkok') if true).
statement(hotelOnto, 'f'@'hasAddress'('f'@'Column Bangkok','48 Sukhumvit Soi 16, 10110 Bangkok') if true).
statement(hotelOnto, 'f'@'proximateAirport'('f'@'Column Bangkok','f'@'Suwanabumme Airport') if true).
statement(hotelOnto, 'f'@'hasActivities'('f'@'Column Bangkok',['Fitness Centre', 'Outdoor Swimming Pool']) if true).
statement(hotelOnto, 'f'@'nearBy'('f'@'Column Bangkok','f'@'Asoke BTS Station') if true).
statement(hotelOnto, 'f'@'degreeOfComfort'('f'@'Column Bangkok',8.4) if true).
statement(hotelOnto, 'f'@'degreeOfServices'('f'@'Column Bangkok',7.9) if true).
statement(hotelOnto, 'f'@'degreeOfStaff'('f'@'Column Bangkok',7.7) if true).

statement(hotelOnto, 'f'@'roomType'('f'@'CBGuest1','Single Room') if true).
statement(hotelOnto, 'f'@'hasPrice'('f'@'CBGuest1',156) if true).
statement(hotelOnto, 'f'@'hasView'('f'@'CBGuest1','City') if true).
statement(hotelOnto, 'f'@'roomType'('f'@'CBGuest2','Double Room') if true).
statement(hotelOnto, 'f'@'hasPrice'('f'@'CBGuest2',257) if true).
statement(hotelOnto, 'f'@'hasView'('f'@'CBGuest2','City') if true).



%-------------------------------------------------
statement(hotelOnto, 'f'@'hasRatingStar'('f'@'InterContinental Bangkok',5) if true).
statement(hotelOnto, 'f'@'numberOfRoom'('f'@'InterContinental Bangkok',381) if true).
statement(hotelOnto, 'f'@'hasRoom'('f'@'InterContinental Bangkok','f'@'InGuest1') if true).
statement(hotelOnto, 'f'@'hasRoom'('f'@'InterContinental Bangkok','f'@'InGuest2') if true).
statement(hotelOnto, 'f'@'hasLocation'('f'@'InterContinental Bangkok','Bangkok') if true).
statement(hotelOnto, 'f'@'hasAddress'('f'@'InterContinental Bangkok','973 Ploenchit Road, 10330 Bangkok') if true).
statement(hotelOnto, 'f'@'proximateAirport'('f'@'InterContinental Bangkok','f'@'Suwanabumme Airport') if true).
statement(hotelOnto, 'f'@'degreeOfComfort'('f'@'InterContinental Bangkok',9.5) if true).
statement(hotelOnto, 'f'@'degreeOfServices'('f'@'InterContinental Bangkok',9.3) if true).
statement(hotelOnto, 'f'@'degreeOfStaff'('f'@'InterContinental Bangkok',9.5) if true).


statement(hotelOnto, 'f'@'roomType'('f'@'InGuest1','Double Room') if true).
statement(hotelOnto, 'f'@'hasPrice'('f'@'InGuest1',260) if true).
statement(hotelOnto, 'f'@'hasView'('f'@'InGuest1','City') if true).
statement(hotelOnto, 'f'@'roomType'('f'@'InGuest2','Deluxe Double') if true).
statement(hotelOnto, 'f'@'hasPrice'('f'@'InGuest2',347) if true).
statement(hotelOnto, 'f'@'hasView'('f'@'InGuest2','River') if true).

%-------------------------------------------
meta_statement(hotelOnto, 'rdf'@'type'('f'@'Amari Boulevard Bangkok','f'@'Hotel') if true).
meta_statement(hotelOnto, 'rdf'@'type'('f'@'Column Bangkok','f'@'Hotel') if true).
meta_statement(hotelOnto, 'rdf'@'type'('f'@'InterContinental Bangkok','f'@'Hotel') if true).
meta_statement(hotelOnto, 'rdf'@'type'('f'@'ABBGuest1','f'@'Room') if true).
meta_statement(hotelOnto, 'rdf'@'type'('f'@'ABBGuest2','f'@'Room') if true).
meta_statement(hotelOnto, 'rdf'@'type'('f'@'CBGuest1','f'@'Room') if true).
meta_statement(hotelOnto, 'rdf'@'type'('f'@'CBGuest2','f'@'Room') if true).
meta_statement(hotelOnto, 'rdf'@'type'('f'@'InGuest1','f'@'Room') if true).
meta_statement(hotelOnto, 'rdf'@'type'('f'@'InGuest2','f'@'Room') if true).
```

Figure I.4 MP and MMP of hotel ontology

```
meta_statement('rdf'#'type'('f'#H,'f'#'Cheap_Hotel') if
    'f'#'hasRoom'('f'#H,'f'#R) and 'f'#'roomType'('f'#R,'Double Room') and
    'f'#'hasPrice'('f'#R,Pc) and 'swrlb'#'builtinAtom'('lessThanOrEqual', Pc, 120)).

meta_statement('rdf'#'type'('f'#H,'f'#'Reasonably_Priced_Hotel') if
    'f'#'hasRoom'('f'#H,'f'#R) and 'f'#'roomType'('f'#R,'Double Room') and
    'f'#'hasPrice'('f'#R,Pc) and 'swrlb'#'builtinAtom'('greaterThan', Pc, 120) and
    'swrlb'#'builtinAtom'('lessThan', Pc, 200)).

meta_statement('rdf'#'type'('f'#H,'f'#'Expensive_Hotel') if
    'f'#'hasRoom'('f'#H,'f'#R) and 'f'#'roomType'('f'#R,'Double Room') and
    'f'#'hasPrice'('f'#R,Pc) and 'swrlb'#'builtinAtom'('greaterThanOrEqual', Pc, 200)).

meta_statement('rdf'#'type'('f'#H,'f'#'Big_Hotel') if
    'f'#'numberOfRoom'('f'#H,N) and 'swrlb'#'builtinAtom'('greaterThanOrEqual', N, 200)).

meta_statement('rdf'#'type'('f'#H,'f'#'Medium_Hotel') if
    'f'#'numberOfRoom'('f'#H,N) and 'swrlb'#'builtinAtom'('lessThan', N, 200) and
    'swrlb'#'builtinAtom'('greaterThan', N, 150)).

meta_statement('rdf'#'type'('f'#H,'f'#'Small_Hotel') if
    'f'#'numberOfRoom'('f'#H,N) and 'swrlb'#'builtinAtom'('lessThanOrEqual', N, 150)).

statement('f'#'betterThan'('f'#H1, 'f'#H2) if
    'f'#'hasAverageScore'('f'#H1, A1) and 'f'#'hasAverageScore'('f'#H2, A2) and
    'swrlb'#'builtinAtom'('greaterThan', A1, A2)).
```

Figure I.5 Rules expressed in meta-programs

I.5 Implementation of the Meta-interpreter

The meta-interpreter is implemented in WIN-PROLOG. It is a simple Vanilla meta-interpreter, the meta-interpreter can reason with meta-programs transformed from an SW ontology. We show the Prolog code of the meta-interpreter in Figure I.6.

```
:- op(888, xfy, and).
:- op(999, xfx, if).
:- op(555, yfy, #).
:- op(666, xfx, @).


demo(true) :-!.
%-----------------------------
demo(A and B) :- demo(A),demo(B).
%-----------------------------
demo(Question) :-
        statement(Question if SubQuestion),
        demo(SubQuestion).

%-----------------------------
demo(Question) :-
        meta_statement(Question if SubQuestion),
        demo(SubQuestion).

%-----------------------------
demo(Question) :-
        axiom(Question if SubQuestion),
        demo(SubQuestion).
%-----------------------------
demo(built_in(BI)) :- BI.
```


Figure I.6 Implementation of the Meta-interpreter

I.6 Hotel Information System

Our hotel information system is operating under a website to allow users to query information about hotels. The information is represented by in a SW ontology with rules. This ontology is then transformed into meta-programs in WIN-PROLOG. We deployed our system in ProWeb Server in order to link HTML page with the meta-interpreter, and meta-programs. Sequence diagram of a query function of our system is depicted in Figure I.7.

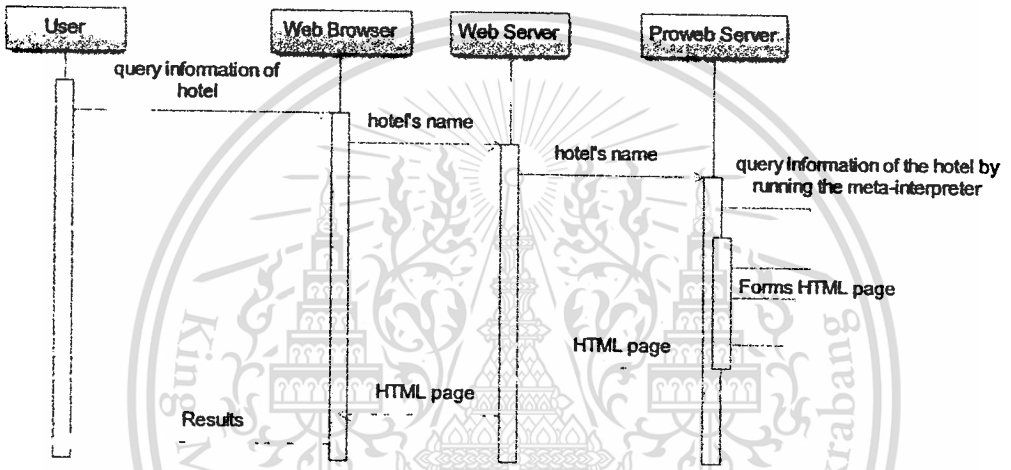


Figure I.7 Sequence diagram of Hotel Information System

Firstly, user puts a name of a hotel to query information of it, the hotel's name will pass to Web Server and to the Proweb Server. At Proweb Server, with this given hotel's name, queries for information of the hotel are made via demo predicate as shown in Figure I.9, the meta-interpreter then reasons with the meta-program to derive answers. After that, Proweb Server forms HTML page to display the answers. This HTML page will be returned to Web Server and Web Browser. Finally, The Web Browser will show the results for the user.

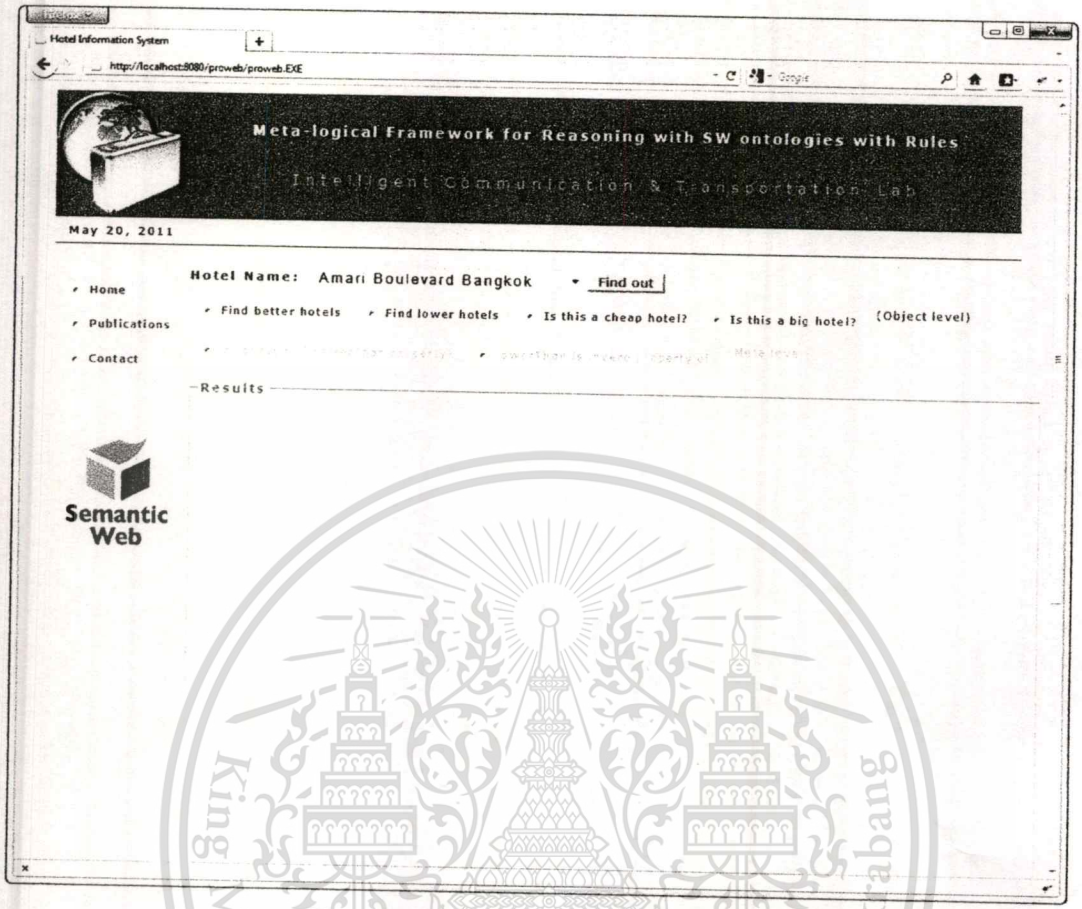


Figure I.8 Our Hotel Information System

Figure I.8 shows the home page of our Hotel Information System. From this page, chose hotel name in the box and click 'Find out' button, the information of the hotel with the chosen name will be shows in Results field as in Figure I.9 below.

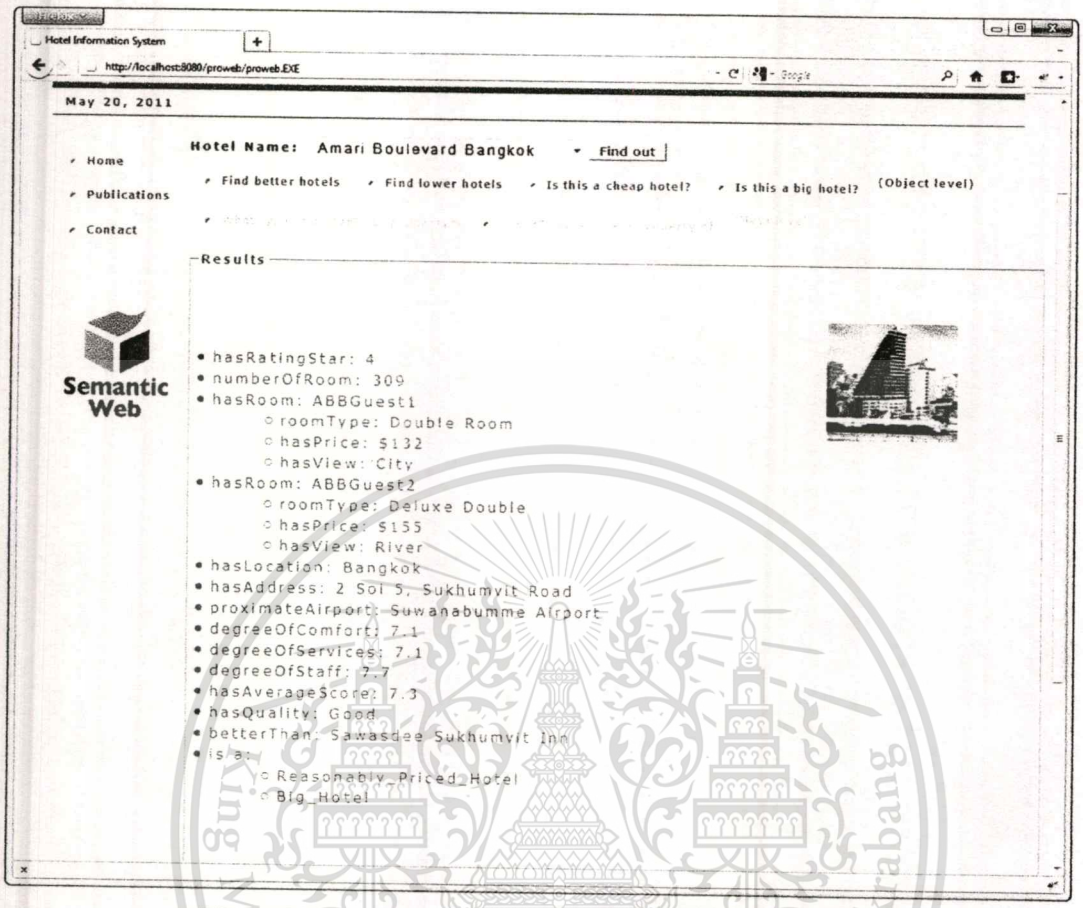


Figure I.9 Query answering of the Hotel Information System

Figure I.9 shows the results when we query information about a hotel namely "Amari Boulevar Bangkok". To give these results, our system will pose queries via demo predicate as in Figure I.10, and then meta-interpreter will reason with the meta-programs to answer the queries.

```
demo('f'#'hasRatingStar'('f'#HotelName, N)).
demo('f'#'numberOfRoom'('f'#HotelName, N)).
demo('f'#'hasRoom'('f'#HotelName, 'f'#Room)).
demo('f'#'hasLocation'('f'#HotelName, Location)).
demo('f'#'hasAddress'('f'#HotelName, Address)).
demo('rdf'#'type'('f'#HotelName, 'f'#HotelType)).
...

demo(P('f'#HotelName, S)).
```

Figure I.10 Queries for hotel information

Our system also supports for querying better hotels, worse hotels or querying whether this hotel is cheap or not, etc. For example, when we click the "Find better hotels" button, the names of hotels which are better than the chosen hotel are listed in Results field as shown in Figure I.10. These answers are given by using the rule "`betterThan`" as we showed in Figure I.5.

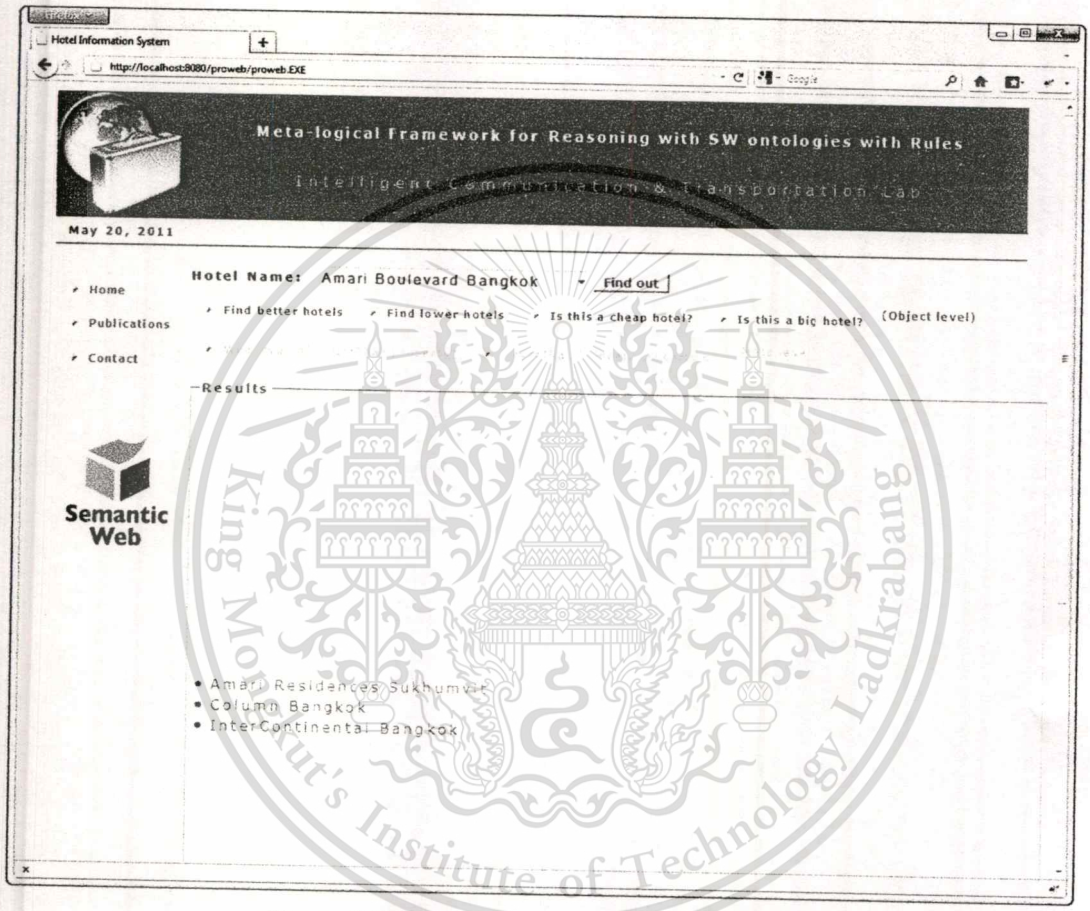


Figure I.11 Results for query "Find better hotels"

Appendix II Author Biography

1	**Personal Information**						
1.1	First Name	TRANG					
	Family Name	MAI XUAN					
1.2	Nationality	Vietnamese					
1.3	Date of Birth	Date	06	Month	Aug	Year	1985
1.4	Gender	Male	√	Female			

2	**Education**				
2.1	**Bachelor's Degree**				
2.1.1	Program	Bachelor of Computer Engineering			
2.1.2	Faculty	Faculty of Information Technology			
2.1.3	University	Hanoi University of Technology, Hanoi, Vietnam			
2.1.4	Year of Start	Month	Sep	Year	2003
2.1.5	Year of Completion	Month	Aug	Year	2008
2.2	**Master's Degree**				
2.2.1	Program	Master of Computer Engineering			
2.2.2	Faculty	International College			
2.2.3	University	King Mongkut's Institute of Technology Landkrabang, Bangkok, Thailand			
2.2.4	Year of Start	Month	Jun	Year	2009
2.2.5	Year of Completion	Month	May	Year	2011

3	**Employment Record**

- Worked as an internship at the end of my bachelor study for 5 months in 2007 and as a software engineer in EVSoft Company for 12 months. The general director was Dr. Pham Huy Hoang.

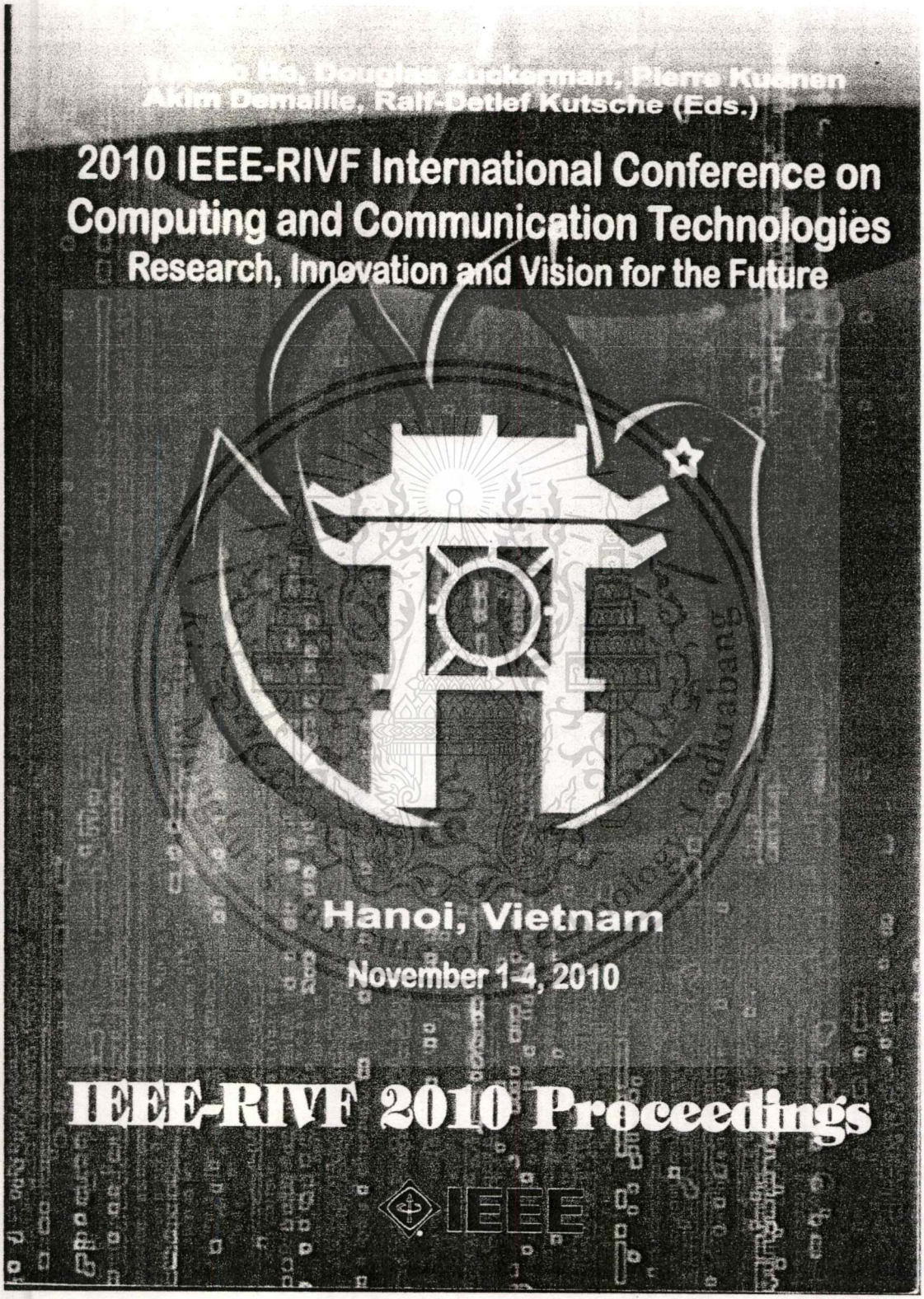
- Worked in some projects such as CTMS Framework (Credit Training Management System), EPM (Entire Project Monitoring) a system run on web environment for managing projects.

4	**Publications**

1. A Meta-logical Approach for Reasoning with an OWL 2 Ontologies. In *proc. of the 2010 IEEE-RIVF International Conference on Computer Sciences: Research, Innovation & Vision for the Future*, November 01-04, 2010, Hanoi, Vietnam, pp. 35-40.

2. A Meta-logical Approach for Reasoning with SWRL Ontologies. In *proc. of the IMECS 2011 International MultiConference of Engineers and Computer Scientists*, Hong Kong, March 16-18, 2011, pp. 112-117.

3. A Meta-logical Framework for Reasoning with Semantic Web Ontologies with Rules in OWL 2. This paper is accepted by WCE 2011, and is going to appear in *proc. of the WCE 2011 World Congress on Engineering 2011*, London, U.K, July 6-8, 2011.



A Meta-logical Approach for Reasoning with an OWL 2 Ontology

Visit Hirankitti
School of Computer Engineering,
Faculty of Engineering, King Mongkut's Institute of Technology Ladkrabang,
Ladkrabang, Bangkok 10520, Thailand
khvisit@kmitl.ac.th

Trang Mai Xuan
International College,
King Mongkut's Institute of Technology Ladkrabang,
Ladkrabang, Bangkok 10520, Thailand
trangmx@gmail.com

***Abstract*—A recent development of a Semantic Web language is OWL 2, an extension of OWL. So far OWL 2 has been designed by W3C as the language for representing a web ontology. Earlier we have developed a meta-logical approach for reasoning with Semantic Web ontologies expressed in OWL [6], with the new extension—OWL 2, in this paper we shall extend our previous work to support this richer language. A meta-interpreter, defined by a demo(.) predicate together with auxiliary axioms, is proposed and used for reasoning with OWL 2 ontologies expressed in terms of meta-programs. Finally we demonstrate some expressiveness of the meta-interpreter.**

Keywords-Semantic Web; Ontologies; Meta-logic; Logic Programming.

I. INTRODUCTION

Ontology is an essential part of Semantic Web (or briefly 'SW') as it forms vocabulary and statements for representing knowledge shared across the web. Ontology can be processed automatically or reasoned logically by computers. Several XML-based markup languages have been developed for expressing an ontology, they were influenced by different formalisms such as logic, and description logic [1]. For example, a core data representation format for SW is RDF [2] which represents resources in the form of subject-predicate-object triples; RDF Schema (RDFS), created together with its formal semantic within RDF, is used to describe classes, properties and their relationships and we use them both to create a lightweight ontology: OWL [5] is a language derived from description logic, and offers more constructs over RDFS. OWL is used to create a more expressive ontology. OWL has been successfully applied to many problems in computer science, and has rapidly become a de facto standard for ontology development and data interchange. Despite its success, some numerous contexts, in which the language has been applied to, have revealed its deficiencies. Therefore, to overcome that, an improvement of OWL, namely 'OWL 2,' has been designed by the W3C Working Group [4].

In our previous work [6], we developed a meta-logical approach for reasoning with SW ontologies described in RDF, RDFS and OWL. To extend it to cope with OWL 2, in this paper we enhance its framework to make it more expressive in order to reason with ontologies described in OWL 2.

This research is sponsored by the Japan International Cooperation Agency (JICA) under the ASEAN University Network / Southeast Asia Engineering Education Development Network (AUN SEED-Net) program

The rest of the paper is organized as follows. In the next section we first give an overview of OWL and address its limitations. In section III we investigate new features of OWL 2 which overcome the limitations in OWL. Accordingly, we extend our previous work to reason with OWL 2 in Section IV by introducing new auxiliary axioms for the interpreter, and after that in Section V we demonstrate how the meta-interpreter together with those axioms can be used to reason with an OWL 2 ontology. We discuss related works in section VI. Finally, in section VII we conclude this work.

II. OWL AND ITS LIMITATIONS

A. OWL Overview

Semantic Webs are designed to support automated processes and intelligent agents, so that they can access and process semantic information automatically. Semantic information, in the form of ontologies, is defined as well-formed constructs to represent concepts in a certain domain, so that intelligent agents can interpret their meaning logically. To serve this purpose, many ontology languages such as RDF, RDFS and OWL were developed. RDF (Resource Description Framework) provides a basis data model for SW. However, RDF itself does not support a data schema. This led to the development of RDFS which provides facilities to define simple taxonomics among concepts and relations. Whilst RDFS is used to represent a simple ontology, a more expressive language for ontology representation, namely 'OWL' (Web Ontology Language), was also developed.

OWL has become a new standard language for describing an ontology. It supports several features beyond the simple definition of the hierarchies of RDFS (using rdfs:subProperty, rdfs:subClassof); in OWL we can define relations between properties and classes. More expressively, OWL allows properties to be transitive, symmetric, inverse, functional, and inversely functional. OWL also supports an ability to define complex classes in terms of logical combinations (e.g. union) of other classes. Furthermore, in OWL we can state which objects (individuals) belong to which class, and what the property values are for the specific individuals. Equivalence properties can be asserted on classes and properties, disjoint properties can be asserted on classes, as well as equality and inequality properties can be asserted between individuals.

978-1-4244-8072-2'10 $26.00 ©2010 IEEE

Although OWL is an expressive language for ontology representation, it is far from being complete.

B. Limitations in OWL

Practicality has revealed OWL's lack of several necessary constructs in both aspects of syntax and expressiveness. In the syntactic aspect, due to the missing of some constructs to deal with patterns in OWL, in some cases we have to represent knowledge in a verbose manner. For example, OWL allows to state that two subclasses are disjoint, but in order to represent that *several subclasses are pairwise disjoint*, we have to adopt many axioms, each states that the two subclasses are disjoint, this is obviously not very concise.

Another drawback is whilst OWL provides means to assert values of a property for an individual and also to express that two individuals are related by some properties, it does not provide constructs for asserting values that an individual does not have, and also does not provide constructs for expressing that two individuals are not related by some properties.

A major drawback of OWL was also revealed in its limitation to deal with properties. This is due to the fact that whilst OWL includes a relatively rich set of class constructors, its treatment on properties is rather weak, i.e. it lacks of constructs for expressing additional restrictions on properties. In other words, OWL limits the characteristics of properties, i.e. although it is possible to declare properties are transitive, symmetric, functional or inversely functional, it does not allow us to declare a property is asymmetric, reflexive, or irreflexive. A definition of disjoint properties is that some two individuals cannot be connected by two properties which are disjoint, unfortunately this kind of definition is not supported by OWL. In addition, we can define a new class as being a composition of other classes in OWL, but we cannot do so with properties. All these limitations can be overcome by OWL 2, which was developed after OWL by W3C [4].

III. OWL 2 – The New OWL

OWL 2 goes beyond the original OWL by adding to it some new features in order to solve the problems of syntax and expressiveness in OWL. Since these new features will be dealt with later by our framework, we shall explain here how the new features were added to the language, so that you can see shortly how our framework can deal with them.

A. Syntactic Sugar

As referred to in the previous section, some patterned knowledge is difficult to express in OWL. To overcome it, OWL 2 adds new constructs to make some common patterns easier to express. For instance, to declare that multiple classes are pairwise disjoint, with OWL several axioms are needed, whereas in OWL 2 we need only one axiom to do by using owl:AllDisjointClasses constructor. In OWL 2 we can also use just owl:disjointUnionOf to define a class as a union of other classes, all of which are pairwise disjoint. Another issue is that sometimes we want to state that two individuals are not related by a property, or to assert values that an individual does not have, OWL 2 provides a means to do that by using owl:NegativePropertyAssertion construct.

B. New Constructs for Properties

Apparently, OWL constructs were defined based on DL SHOIN, this led to major drawbacks regarding expressiveness of property declaration as said earlier. OWL 2, however, was designed based on DL SROIQ [9] by including new constructs for expressing additional restriction on properties, new characteristics of properties, incompatibility of properties, and property chains and keys. These new constructs significantly enhance the expressiveness of the language.

In addition, some constructors were added to OWL 2 for declaring that properties are reflexive, irreflexive, or asymmetric. owl:asymmetricProperty defines a rule is asymmetric if A is related to B via this rule, but B is never related to A via this rule. Reflexive property is defined by owl:reflexive, meaning that every individual A is related to itself via such a rule. In addition, irreflexive property is defined by owl:irreflexive.

Moreover, to increase OWL 2's relational expressiveness, some constructs for defining relationships among properties were introduced. OWL 2 provides owl:propertyDisjointWith and owl:AllDisjointProperties to declare disjoint properties. OWL 2 also provides owl:propertyChainAxiom to express property chains, i.e. propagation of one property to another.

Another feature in OWL 2 is an ability to express key constraints on data properties, which is a core feature of database technology. OWL 2 provides key axioms in the form of Haskey(C $P_1 \ldots P_n$) to state that the object, or data, properties P_i are keys for named instances of class C, i.e. no two named instances of C coincide on all values of all the properties P_i. Table I briefs some differences between OWL and OWL 2.

Features	OWL	OWL 2
Description logic	based on DL SHOIN	based on DL SROIQ
Disjoint Classes	owl:disjointWith: two classes are disjoint	owl:AllDisjointClasses: several classes are pairwise disjoint.
Property characteristics	transitive, symmetric, functional	Extra-property: asymmetric, reflexive, irrefexive
Property chains	not support	owl:propertyChainAxiom
Negated property assertion	not support	owl:NegativePropertyAssertion
Key constraints	not support	owl:hasKey

Table I. Comparison between OWL and OWL 2

IV. Our Approach

To handle new features in OWL 2, in this section we extend our previous framework [6] so that it can reason with an OWL 2 ontology.

A. Our Framework

In the previous work we have argued that SW languages and a logic programming language were similar in that they are both *declarative*. To adopt logic programming for SW, in [14] information of an SW ontology is initially transformed into a prolog program and then a prolog interpreter is used to derive answers for a query. Alternatively, in our approach we applied logic programming in the context of meta-logic [10] to

SW. We have developed an agent-like meta-interpreter to reason with SW ontologies in the form of meta-programs [6]. This agent can reason with multiple distributed SW ontologies [8], can communicate SW information among agents [7], and can provide SW services to other agents [5].

Our framework forms a logical system consisting of meta-programs and an inference engine. The former is in the form of logical sentences representing an SW ontology at the meta-level. That is, an ontology described by OWL 2 is transformed into a meta-logical representation. The later 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 can 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 obtain an SW resource like a web browser.

Figure 1. Our meta-logical system

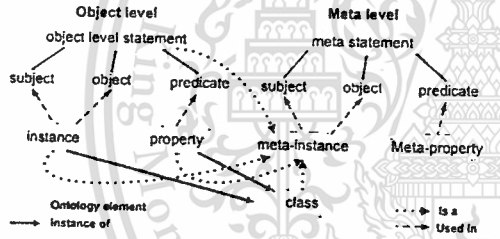


Figure 2. Ontology elements at the object level and meta level

To explain our framework, in the next three sub-sections we first introduce our meta-language used for formulating the meta-programs of SW ontologies, and then explain the meta-programs in details. Finally we describe our meta-interpreter. SW ontologies, which our framework reasons with, are described by OWL 2 which includes the new features referred to in the previous section.

B. Meta-languages 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 depicted in Figure 2. At the object level, an instance can be an individual or a literal of a domain, e.g. `'john'`, and property is a relationship between individuals, or is an individual's attribute, e.g. `'hasSon'`, `'type'`. At the meta-level, a meta-instance can be an individual, a property, a class, or an object-level statement. A meta-property is a property to describe a meta-instance's attribute or a relationship between/among meta-instances, e.g. `'reflexive'`, `'disjointWith'`.

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>`, such as `'f':'son'`, `'f':'hasSon'`, `'owl':'reflexive'`, etc. Henceforth this qualified name will be used throughout.

According to our framework, in an SW ontology we distinguish between its object and meta levels, and similarly its object and meta languages. The object language specifies objects and their relationships in the real world. The meta-language describes the syntactic form of the object language. Hence, we have formulated two meta-languages: one discussing mainly about objects and their relationships we call it "meta-language for the object level (**ML**)", and the other we call "meta-language for the meta-level (**MML**)," which discusses mainly about classes, instances, properties and their relationships.

- **Meta-language for the object level (ML)**

Objects and their relationships at the object level are specified in an SW ontology and this information is expressed by the elements of ML below.

Meta-constant specifies a name of an object and a literal, e.g. `'son'`, including a reference, e.g. a namespace, the latter is a meta-constant of MML. This means that ML and MML are not totally 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 objects, or a name of an object's property—i.e. an object-level predicate name, such as `'hasSon'`, `'name'`. It also stands for other meta-level function symbol, e.g. `'←'`, `'∧'`, `':'`.

Meta-term is either a meta-constant or a meta-variable or meta-function symbol applied to a tuple of meta-terms, e.g. `'f':'hasSon'`, `'owl':'reflexive'`. To express 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-variable, e.g. `'f':'hasSon'('f':'fa','f':'son')`; for a negative predicate ¬P, it has the form negative(P,S,O).

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':'hasSon'('f':'fa','f':'son') ← true).
```

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

Apart from the object language, an SW ontology also defines classes, properties, their relationships, as well as class-instance relations, and we argue that this information is *meta-information of the object level*. Here we express this information by MML which 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 `':'`; or a name of set operators applied on

classes such as union; or a meta-predicate name being a name of a relation between entities; or a name of characteristic of a property, which may fall into one of the following categories:

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

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

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

Class-property relations: keys, etc.

Relations between literals and instances/classes/properties: we can take these relations as attributes of instances, of classes, or of properties, e.g. comment, label.

Characteristics of properties: reflexive, asymmetric, etc.

Meta-term being either a meta-constant or a meta-variable or meta-function symbol applied to a tuple of meta-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. For a negative predicate, it has the form ¬`Pred`, where ¬ is a function symbol.

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':'propertyDisjointWith'(P,DP) ←
  'owl':'propertyDisjointWith'(DP,P).
```

Meta-statement being a meta-predicate or meta-predicates connected by logical connective. It has two forms *meta_statement(meta-level-sentence)* and *axiom(meta-level-sentence)*, the latter represents a rule for a mathematical axiom, e.g:

```
meta_statement('owl':'propertyDisjointWith'
      ('f':'likes','f':'dislikes') ← true).
axiom('owl':'propertyDisjointWith'(P,DP) ←
  'owl':'propertyDisjointWith'(DP,P)).
```

C. *Meta-programs of an SW Ontology*

Each SW ontology is transformed into a meta-program containing a (sub-)meta-program expressed in ML, called **MP**, and a (sub-)meta-program expressed in MML, called **MMP**. Another meta-program expresses some mathematical axioms for classes and properties in MML called **AMP** is also needed for the inference engine to reason with **MP** and **MMP**.

- **Meta-program for the object-level (MP)**

MP contains meta-statements for the object level in the forms of `statement(P(S,O) ← true)`. This meta-program expresses information about instances and their relationships. In order to work with *negative* property assertion in OWL 2, we use a meta-statement form `statement(NP(P,S,O) ← true)`, where `NP` expresses negative property. For example, to declare person M01 is not the father of person M03 we have a meta-statement in **MP** as follows:

```
statement('owl':'negativeProperty'
  ('f':'fatherOf','f':'M01','f':'M03') ← true).
```

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

MMP contains meta-statements for classes, properties, their relationships, and class-instance relations in terms of meta-rules. The **MMP** is represented in the forms as follows:

```
meta_statement(P(S,O) ← true),
meta_statement(P(S,Os) ← true),
meta_statement(C(Prop) ← true).
```

where `P`, `S`, `O` are predicate, subject, and object of a triple (S, P, O) defined in the ontology. `C` is a characteristic of a property `Prop`. `Os` is a tuple composing of several objects.

The new features in OWL 2 are transformed into **MMP** with the following new meta-statements.

New meta-statements for syntactic sugar:

```
meta_statement(
    'owl':'AllDisjointClasses'(Cs) ← true).
// All classes in Cs are pairwise disjoint.
meta_statement(
  'owl':'disjointUnionOf'(C,Cs) ← true).
// Class C is the disjoint union of classes Cs.
```

New meta-statements for expressing new constructs for properties:

```
meta_statement(
  'owl':'propertyDisjointWith'(P,DP) ← true).
// Properties P and DP are disjoint.
meta_statement(
  'owl':'AllDisjointProperties'(Ps) ← true).
// Properties in Ps are pairwise disjoint.
meta_statement(
  'owl':'propertyChainOf'(P,[P1,P2]) ← true).
// Property P is composition of properties P1 and P2.
meta_statement('owl':'reflexive'(P) ← true).
meta_statement('owl':'irreflexive'(P) ← true).
meta_statement('owl':'asymmetric'(P) ← true).
// Property P is reflexive, irreflexive, or asymmetric.
meta_statement('owl':'hasKey'(C,Ps) ← true).
// Properties in Ps are keys of named instances of class C.
```

- **Meta-program for the axioms (AMP)**

AMP contains axioms for classes and properties, they are expressed in the meta-rule form. In [6], we adopted many axioms to support OWL. For OWL 2 however, we need *more axioms* to handle its new features. *These axioms are related to new constructs and characteristics of property:*

```
axiom('owl':'propertyDisjointWith'(P,DP) ← (asdp)
  'owl':'propertyDisjointWith'(DP,P)).
// property disjoint is symmetric.
axiom(P(S,O) ←                                  (acdp)
    'owl':'propertyDisjointWith'(P,DP) ∧
    DP(S,O1) ∧ 'owl':'differentFrom'(O,O1)).
Properties P and DP are disjoint.
axiom(P(S,O) ←                                  (acap)
  'owl':'asymmetric'(P) ∧ P(O,S1) ∧
```

```
'owl':'differentFrom'(S,S1)).
```
// Property P is asymmetric.

```
axiom(P(S,S) ← 'owl':'reflexive'(P)).                    (acrep)
```
// Property P is reflexive.

```
axiom(P(S,O) ← 'owl':'irreflexive'(P) ∧                  (acirp)
        'owl':'differentFrom'(S,S1)).
```
// Property P is irreflexive.

```
axiom(P(S,O) ←                                           (acpc)
  'owl':'propertyChainOf'(P,[P1,P2])
  P1(S,O1) ∧ P2(O1,O)).
```
// Axiom to handle property chain.

```
axiom('owl':'hasKey'(C,P) ←                              (akeyp)
  'rdf':'domain'(P,C) ∧
  P(S1,O) ∧ P(S2,O) ∧ 'owl':'sameAs'(S1,S2)).
```
// P is a key of a named instance of class C.

```
axiom('¬'P(S,O) ←                                        (anap)
  'owl':'negativeProperty'(P,S,O)).
```
// Axiom to handle a negative property.

D. The Meta-interpreter

The meta-interpreter in our framework is built to reason with the meta-programs **MPs**, **MMPs**, and **AMPs** and can be used to develop an intelligent agent to reason with SW ontologies. It is defined by a `demo` predicate of the form `demo(A)`. With this predicate the meta-interpreter can infer an answer A from the meta-programs. The interpreter is defined by adapting the Vanilla meta-interpreter [10] for reasoning with the meta-programs, which 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, and (3) `axiom(A← B)` for a mathematical axiom. The definition of `demo/1` is:

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

The first clause (true) is the basic case for proving an atom `true`. The second clause (conj) is used for proving a conjunction goal. Three last clauses (ost), (mst), and (ast) are used for proving three meta statements from the three meta-programs **MP**, **MMP**, and **AMP** respectively.

V. Query Answering with the Meta-Interpreter

With our framework SW ontologies are transformed into meta-programs **MP** and **MMP**. The meta-programs are inputs to the meta-interpreter and they are all implemented in Prolog. Then the meta-interpreter is used to derive conclusions from the SW ontologies.

The family ontology [11] is an example to test the new features in OWL 2. This ontology, however, does not make use of all new features in OWL 2. In order to test all the new features, we thus add more statements to this ontology. After it is transformed into meta-programs, here is a part of them:

- **The MMP program:**

```
meta_statement('owl':'propertyDisjointWith'              (1)
  ('f':'likes','f':'dislikes') ← true).

meta_statement('owl':'propertyChainOf'                   (2)
  ('f':'hasAncestor',['f':'hasAncestor',
    'f':'hasAncestor']) ← true ).

meta_statement('owl':'reflexive'                         (3)
  ('f':'likes') ← true).

meta_statement('owl':'irreflexive'                       (4)
  ('f':'hasSibling') ← true ).

meta_statement('owl':'asymmetric'                        (5)
  ('f':'hasSon') ← true ).

meta_statement('owl':'hasKey'                            (6)
  ('f':'Person','f':'hasID) ← true).
...
```

- **The MP program:**

```
statement('f':'hasSon'                                   (1')
  ('f':'fa','f':'son') ← true).

statement('f':'hasSibling'                               (2')
  ('f':'son','f':'daughter') ← true).

statement('f':'hasAncestor'                              (3')
  ('f':'son','f':'fa') ← true).

statement('f':'hasAncestor'                              (4')
  ('f':'fa','f':'grandpa') ← true).

statement('f':'likes'                                    (5')
  ('f':'fa','f':'persX') ← true).

statement('f':'dislikes'                                 (6')
  ('f':'fa','f':'persY') ← true).

statement('owl':'negativeProperty'                       (7')
  ('f':'hasSon','f':'fa','f':'persZ') ← true).
...
```

We then pose some queries to the meta-interpreter and get the answers as the following:

```
?- demo('f':'hasSon'('f':'fa',X)).
  X = 'f':'son'.
```
// The adopted clauses are (ost), (1'), (ast), (acap), (conj), (mst), (5) and (true).

```
?- demo('f':'hasSibling'('f':'son',X)).
  X = 'f':'daughter'.
```
// The adopted clauses are (ost), (2'), (ast), (acirp), (conj), (mst), (4) and (true).

```
?- demo('f':'likes'('f':'fa',X)).
   X = 'f':'persX';
   X = 'f':'fa'.
```
// 1st answer is supported by (ost), (5'), and (true), and 2nd answer by (ast), (acrep), (mst), (3) and (true).

```
?-demo('f':'dislikes'('f':'fa',X)).
   X = 'f':'persZ';.
```
// The adopted clauses are (ost), (6'), (ast), (acdp), (conj), (asdp) (mst,) (1) and (true).

```
?- demo('f':'hasAncestor'('f':'son',X)).
   X = 'f':'fa'; X = 'f':'grandpa'.
```
// 1st answer is supported by (ost), (3'), and (true), and 2nd answer by (ast), (acpc), (conj), (mst), (2), (true), (ost), (3'), and (4').

```
?- demo('¬''f':'hasSon'('f':'fa',X)).
   X = 'f':'persZ'.
```
//The adopted clauses are (ast), (anap), (ost), (7'), and (true).

VI. Related Works

Grosof, and Horrocks [13] proposed a form of knowledge representation, called 'Description Logic Program' (DLP), which employs rules and OWL DL (OWL Description Logic) ontologies. Their approach supports bi-directional translation between logical sentences from DLP fragment of Description Logic and logic programs (in logic programming).

Every concept instantiation is mapped into a unary relation with the concept name becoming the name of the relation and the individual name becoming the argument. Every instance-property-instance relationship is mapped into a binary relation. In addition, concepts as well as property constructor statements are converted into rules. The distinctions between this approach and ours are the following.

Firstly this approach was designed to support a subset of DAML+OIL, and provides only a mapping from RDFS and DAML+OIL to logic programs, but does not support the new features in OWL 2. Secondly, this approach has a weakness to represent an ontology in a logic program. For example, to represent the statement "a is union of $b_1, b_2, \ldots, b_n$", it requires n rules to do so, i.e. `a(X):-b1(X)`, ..., `a(X):-bn(X)`. However, in our representation, this requires only one statement: `meta_statement('owl':'unionOf'(a, [b1,...,bn]))` ← `true`

which is more compact.

Vassiliadis et. al. [12] proposed a logic programming approach to reason with OWL 2 ontologies. In their approach an SW ontology is first transformed into a prolog program, then a Prolog interpreter is used to answer to a query about this program. Compared with our approach, firstly this approach does not make a distinction between the object level and the meta level, and does not express an ontology at the meta-level. Secondly, the Prolog programs are processed by the Prolog interpreter, whilst we used a meta-interpreter to reason with the meta-programs expressing an ontology. For example, with the same ontology that we use for the query answering in Section V, according to their approach, the ontology would be transformed into a Prolog program:

```
'f':'hasSon'('f':'fa','f':'son').
'f':'likes'('f':'fa','f':'persX').
...
'f':'likes'(X,X):-'f':'likes'(X,Y).
// likes is a reflexive property
...
```

For every reflexive property, one Prolog rule like above is required, however, with our approach we need only one rule `axiom(P(S,S)` ← `'owl':'reflexive'(P))` in **AMP** to express all these reflexive properties.

In addition, adopting their Prolog program, some queries in Section V, such as `?'f':'likes'('f':'fa',X)` and `?'f':'hasSon'('f':'fa',X)`, when posed to the Prolog interpreter will bring the same answers as we get from our meta-interpreter. However, there are queries, that asking meta-information about an ontology, which their approach cannot give answers to, since their Prolog program can provide answers for only the object-level information about an ontology. A query for meta-information such as a query asking what the relation between `fa` and `persX` is, or a query asking what the characteristics of the property `'f':'likes'` is, we cannot get answers from their Prolog program, but our meta-interpreter can answer to all that as following:

```
?- demo(P('f':'fa','f':'persX')).
    P = 'f':'likes'.
?- demo(P('f':'likes')).
    P = 'owl':'reflexive'.
```

VII. Conclusion

We have presented a meta-logical framework for reasoning with ontologies described in the new Semantic Ontology Language–OWL 2. Our previous framework designed to support OWL has been extended to work with OWL 2 by slightly modifying the meta-language and adding some extra auxiliary axioms to support the meta-interpreter.

References

[1] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P. F. Patel Schneider(Eds.), The Description Logic Hand book: Theory, Implementation, and Applications, 2nd ed. Cambridge, 2007.

[2] W3C The Resource Description Framework. http://www.w3.org RDF

[3] P. F. Patel-Schneider, P. Hayes, I. Horrocks, OWL Web Ontology Language Semantics and Abstract Syntax, W3C Recommendation. http://www.w3.org/tr/2004/rec-owl-semantics-20040210/.

[4] Boris Motik, Peter F. Patel-Schneider, and Bijan Parsia, OWL 2 web ontology language document overview, URL. 2009. http://www.w3.org/TR/owl2-overview .

[5] V. Hirankitti, and V. X. Tran, "Meta-reasoning for Multi-agent Communication of Semantic Web Information," in Int. Journal of Information Technology 3 (2), pp. 91-100, 2006.

[6] V. Hirankitti, and V. X. Tran, "A Meta logical Approach for Reasoning with Semantic Web Ontologies," Proc. of the 4th IEEE Int. Conf. on Computer Sciences: Research, Innovation & Vision for the Future, Vietnam, pp. 228-235, 2006.

[7] V. Hirankitti, and V. X. Tran, "A Meta-logical Approach for Multi-agent Communication of the Semantic Web Information," Proc. of the 16th Int. Conf. on Applications of Declarative Programming and Knowledge Management, Lecture Notes in Computer Science; Vol. 4369, Springer-Verlag, pp. 215-228, 2006.

[8] V. Hirankitti, and V. X. Tran, "Meta-reasoning with Multiple Distributed Ontologies on the Semantic Web," Proc. of the 6th Int. Conf. on Intelligent Technologies, Thailand, pp. 301-309, 2005.

[9] I. Horrocks, O. Kutz, and U. Sattler, "The even more irresistible SROIQ," In: Proceedings of the KR 2006, Lake District, UK, 2006.

[10] 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.

[11] The family ontology, http://www.owldl.com/ontologies/family.owl.

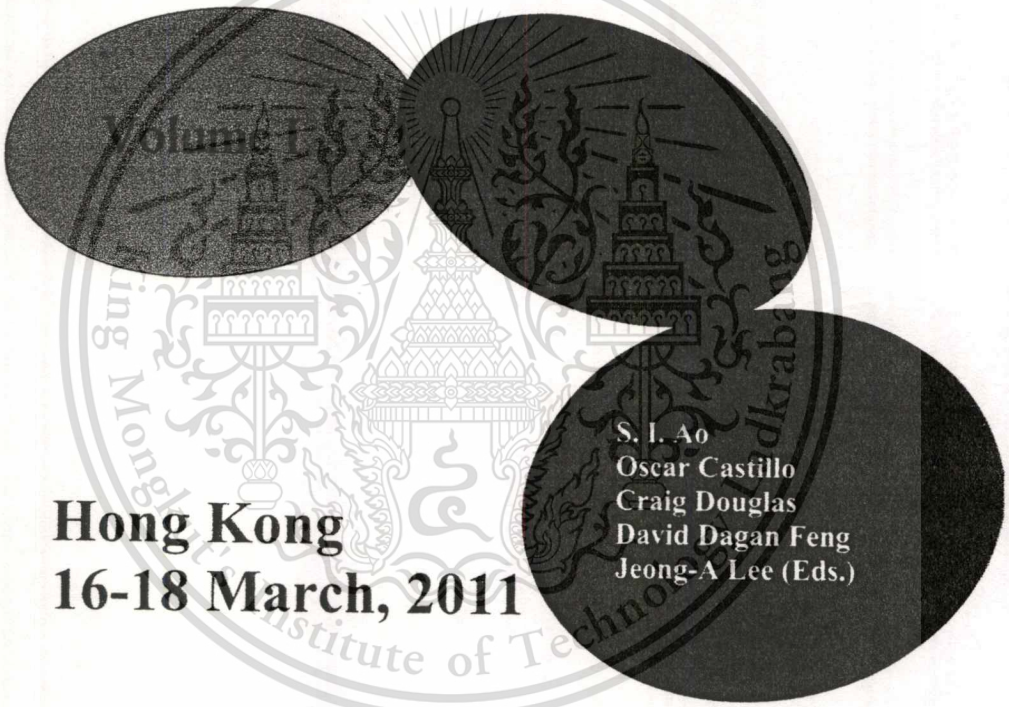
[12] V. Vassiliadis, J. Wielemaker, C. Mungall, "Processing OWL 2 ontologies using Thea: An application of logic programming," Proc. of OWL: Experiences and Direction 2009 (OWLED 2009).

[13] B. N. Grosof, and I. Horrocks, "Description Logic Programs: Combining Logic Programs with Description Logic," In proc. Of the 12th International Conference on the WWW, pp. 45-57 (2003).

Lecture Notes in Engineering and Computer Science

IMECS 2011

International MultiConference of Engineers and Computer Scientists 2011



Hong Kong
16-18 March, 2011

S. I. Ao
Oscar Castillo
Craig Douglas
David Dagan Feng
Jeong-A Lee (Eds.)

IA ENG
International Association of Engineers

ISBN: 978-988-18210-3-4
ISSN: 2078-0958

A Meta-reasoning Approach for Reasoning with SWRL Ontologies

Visit Hirankitti, and Trang Mai Xuan

Abstract—**SWRL is designed for enhancing inferential power on OWL ontologies by introducing rules to the language. With SWRL, rules are allowed to combine with OWL ontologies in order to support deduction on the semantic web ontologies. Earlier we have developed a meta-logical approach for reasoning with Semantic Web ontologies expressed in OWL [5] and OWL 2 [7]; with the advent of SWRL, in this paper we shall extend our framework to support OWL with rules, and hence to support SWRL. Meta-languages together with a meta-interpreter, defined by demo(.) predicate, are proposed and used for reasoning with Semantic Web ontologies with rules.**

Index Terms—**Logic Programming, Meta-logic, Ontology, Rules, Semantic Web.**

I. INTRODUCTION

Ontology is an essential part of the Semantic Web (or shortly "SW") as it forms vocabularies and statements for representing knowledge shared across the web. For an ontology to be processed automatically or reasoned logically by computers, it needs to be specified formally and declaratively. Several XML-based markup languages have been developed for expressing an ontology, they were influenced by different formalisms such as object-oriented approaches, first order logic, and Description Logic [1]. For instance, a core data representation language for SW is RDF [2]; RDF is used to represent resources in a form of subject-predicate-object triples, whereas RDF Schema (RDFS), is used to describe classes, properties and their relationships in the domain discourse, and the RDFS uses them to create a lightweight ontology. Web Ontology Language (OWL) [3] is a language derived from description logic, and offers more constructs over RDFS. OWL is used to create a more meaningful ontology, so that we can infer further information from that ontology. However, OWL still suffers from limitations due to the fact that while the language includes a relatively rich set of class constructors, its ability to describe properties is rather weak. Particularly, in OWL DL, which is based on DL $\mathcal{SHOIN}$, there is no composition constructor, so it is impossible to describe relationships between a composite property and another property. For instance, the composition of the "parent" and "brother" properties cannot be used to define "uncle" property in OWL DL. Therefore, to overcome this limitation the OWL DL needs to be extended. One way to do that is to move from OWL to OWL 2 which is an extension of OWL, and is based on the more expressive Description Logic—DL $\mathcal{SROIQ}$. Another alternative is to extend OWL with a rule language, and SWRL (Semantic Web Rule Language) [4] was proposed for this purpose. SWRL provides Horn clause rule extension to OWL DL, so that an OWL ontology can now be combined with rules. Therefore, SWRL allows deduction to perform on the Semantic Web ontologies.

In our previous work [5], we have developed a meta-logical approach for reasoning with SW ontologies described in RDF, RDFS and OWL. To extend this in order to support rules in the SW ontologies, in this paper we improve our previous framework to support reasoning with SWRL ontologies.

The rest of the paper is organized as follows. In the next section we first give an overview on OWL and SWRL. Next we extend our previous framework to reason with rules in section III, and in section IV we demonstrate how our framework can reason with ontologies with rules. Later we discuss some related work in section V, and finally we conclude our work in section VI.

II. ONTOLOGY LANGUAGES

A. OWL

The Semantic Web has been designed to support automated processes and intelligent agents, so that they can access and process semantic information automatically. Semantic information, in the form of ontologies, is defined as well-formed constructs to represent concepts in a certain domain, so that intelligent agents can interpret their meaning logically. To serve for this purpose, many ontology languages such as RDF, RDFS and OWL were developed. RDF (Resource Description Framework) provides a basis data model for SW. However, RDF itself does not support a data schema. This led to the development of RDFS which provides facilities to define simple taxonomies among concepts and relations. Whilst RDFS is used to represent a simple ontology, a more expressive language for ontology representation, namely "OWL" was later developed.

OWL has become a standard language for expressing an

Visit Hirankitti is with the School of Computer Engineering, Faculty of Engineering, King Mongkut's Institute of Technology Ladkrabang, Ladkrabang Dist., Bangkok 10520, Thailand (phone: 668-0454-9990; e-mail: v_hirankitti@yahoo.com).

Trang Xuan Mai is with the International College, King Mongkut's Institute of Technology Ladkrabang, Bangkok 10520, Thailand (phone: 668-7679-7235, e-mail: trangmx@gmail.com).

ontology. It supports several features beyond the simple definition of the hierarchies of RDFS (using rdfs:subProperty, rdfs:subClassof): in OWL we can define relations between properties and classes. More expressively, OWL allows properties to be transitive, symmetric, inverse, functional, and inversely functional. OWL also supports an ability to define complex classes in terms of logical combinations (e.g. union) of other classes. Furthermore, in OWL we can state which objects (individuals) belong to which class, and what the property values are for the specific individuals. Equivalence properties can be asserted on classes and properties, disjoint properties can be asserted on classes, as well as equality and inequality properties can be asserted between individuals.

Although OWL seems to be a very expressive language for describing an SW ontology, it needs however to embrace rules to allow deduction to perform on the ontology.

B. SWRL

Whist OWL provides rich vocabulary needed for expressing an ontology, but this ontology has limitation on inferential power related to composite properties as mentioned earlier. Therefore, adding rules to OWL will make it a more viable language for ontology representation. The basic idea for OWL rules is to extend OWL with a form of rules while maintaining maximum backward compatibility with OWL's existing syntax and semantics: and RuleML (Rule Markup Language) [8] was adopted as the language to express such a rule. Later, a new language SWRL, which is the result of combining OWL DL—the sublanguage of OWL—and RuleML, was introduced.

SWRL rules are in the form of implication consisting of an antecedent and a consequent, where description-logic expressions can occur in both. The intended interpretation of the rule corresponds to that in the classical first-order logic, that is to assert the rule, whenever the conditions specified in the antecedent hold, then the conditions specified in the consequent must also hold: so an SWRL rule is in this form:

Antecedent → *Consequent*.

Both antecedent and consequent of a rule consist of one or more *atoms*. With homogeneous combination of OWL and RuleML, the *atoms* can be in the form of either C(x), P(x,y), Q(x,z), sameAs(x,y), differentFrom(x,y), or builtIn(*pred*,z_1,....,z_n), where C is an OWL DL description, P is an OWL DL *individual-valued* property, Q is an OWL DL *data-valued* property, *pred* is built-in relation, x and y are either *individual-valued* variables or OWL individuals, and z, z_1, ... z_n are either *data-valued* variables or OWL data literals.

For example, a rule for expressing "uncle" relation can be written as follows:

hasParent(x,y) ∧ *hasBrother*(y,z) → *hasUncle*(x,z).

A rule with multiple atoms in consequent can be transformed into multiple rules. That is, let the multiple atoms in the antecedent form a conjunction *B*, and multiple atoms in consequent form a conjunction $H_1 \wedge H_2$. We can equivalently express one rule of the form of $B \rightarrow H_1 \wedge H_2$ by the two rules $B \rightarrow H_1$ and $B \rightarrow H_2$ (due to $B \rightarrow H_1 \wedge H_2 \equiv B \rightarrow H_1 \quad B \rightarrow H_2$).

SWRL abstract syntax was defined by adding axioms to OWL semantics and its abstract syntax [9] in order to allow the rule axioms, and the syntax of the rule axiom is:

axiom ::=rule
rule ::= 'Implies('[annotation,] antecedent consequent)'
antecedent ::= 'Antecedent('[atom,]')'
consequent ::= 'Consequent('[atom,]')'.

For example, the rule "uncle" can be included to an ontology by adding the following rule axiom to that ontology.

Implies(Antecedent(hasParent(x,y) hasBrother(y,z))
Consequent(hasUncle(x,z))).

SWRL provides a rule extension to OWL, and this allows rules to be represented with OWL ontologies, and these are to be reasoned by our framework in the next section.

III. Our Framework

A. A Meta-logical approach

In our approach we applied logic programming in the context of meta-logic [11] to SW. Our framework forms a logical system consisting of meta-programs and an inference engine. The former is in the form of logical sentences representing a SWRL ontology at the meta-level. That is, an ontology with rules is transformed into a meta-logical representation. The later 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 can be extended to communicate to the Internet to obtain SW ontologies and rules from a web site, communicate with other agents to exchange SW information [6], and draw inference consequences from SW ontologies for the user.

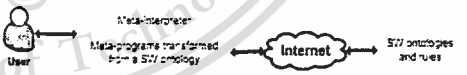


Fig. 1 Our meta-logical system

To explain our framework, in the following sub-sections we first introduce our meta-language used for formulating the meta-programs of SWRL ontologies, and then describe the meta-programs in details. Finally we describe our meta-interpreter.

B. Meta-language for an SWRL 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 depicted in Fig. 2. At the object level, an instance can be an individual or a literal of a domain, e.g. `'John'`, and property is a relationship between individuals, or is an individual's attribute, e.g. `'hasSon'`, `'type'`. At the

meta-level. a meta-instance can be an individual. a property. a class. or an object-level statement. A meta-property is a property to describe a meta-instance's attribute or a relationship between among meta-instances. e.g. `'ref.exive'`. `'disjointWith'`. 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>`. such as `'f':'son'`. `'f':'hasSon'`. `'owl':'reflexive'`. etc. Henceforth this qualified name will be used throughout.

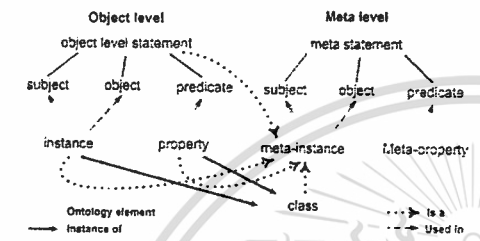


Fig. 2. Object level and Meta level of ontology elements.

According to our framework. in an SW ontology we distinguish between its object and meta information. and similarly its object and meta languages. The object language specifies objects and their relationships in the real world. The meta-language describes the syntactic form of the object language. Since both object-information and meta-information are to be reasoned (and probably manipulated) by a meta-interpreter. only *syntactic forms* are processed by it. Therefore. a meta-representation of the two kinds of meta-statements is required by the meta-interpreter in order to reason with them. This is the reason why both object-information and meta-information statements have to be expressed in yet another meta-program.

Object level information and meta-level information of an SWRL ontology are expressed in our meta-languages. According to these two levels. we classify meta-language elements into two groups: one discussing mainly about objects. their relationships. and SWRL rules. we call it "meta-language for the object level (ML)". and the other we call "meta-language for the meta-level (MML)." which discusses mainly about classes. instances. properties and their relationships.

- **Meta-language for the object level (ML)**

Objects and their relationships at the object level are specified in an SW ontology and this information is expressed by the elements of ML below.

Meta-constant specifies a name of an object and a literal. e.g. `'son'`. including a reference. e.g. a namespace. the latter is a meta-constant of MML. This means that ML and MML are not totally 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 objects. or a name of an object's property—i.e. an object-level predicate name. such as `'hasSon'`. `'name'`. It also stands for other meta-level function symbol. e.g `':-'`. `','`. `';'`.

Meta-term is either a meta-constant or a meta-variable or meta-function symbol applied to a tuple of meta-terms. e.g. `'f':'hasSon'`. or `'owl':'reflexive'` To express 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 variable. e.g. `'f':'hasSon' 'f':'Pa','f':'Son'`). 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-variable appearing in the object-level sentence are universally quantified. One form of this sentence is a Horn-clause. e.g.

```
'f':'hasFather' 'h,F' :- 'f':'hasFather' 'h,F' ,
                         'rdf':'type' o,'f':'Man' .
```

The meta-term. expressing an object-level predicate. is equivalent to a form of Horn-clause with an empty body. Thus. we can put `true` instead of the emptiness in its body. e.g.

```
'f':'hasSon' 'f':'Fa','f':'h' :- true.
```

Meta-statement for the object level reflects an object-level sentence to its existence at the meta-level. It has a form *statementtoobject-level-sentence*. note that this statement is used to represent a Horn-clause rule translated from an SWRL rule. e.g.

```
statement 'f':'hasSon'
  'f':'Fat','f':'h'      :- true .

statement 'f':'hasFather'  F  F  :-
          'f':'hasParent' Chil
          'rdf':'type' F,'f':'
```

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

Apart from the object language. an SW ontology also defines classes. properties. their relationships. as well as class-instance relations. and we argue that this information is *meta-information of the object level*. Here we express this information by MML which 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 `';'`. or a set operator applied on classes such as union: or a meta-predicate name being a name of a relation between entities: or a name of characteristic of a property. which may fall into one of the following categories:

Class-class relations: equivalent class of. disjoint with. etc.
Class-instance relations: instance of. class of. etc.
Property-property relations: subproperty of. chain of. etc.
Class-property relations: keys. etc.
Relations between literals and instances classes properties: we can take these relations as attributes of instances. of classes. or of properties. e.g. comment. label.
Characteristics of properties: reflexive. asymmetric. etc.

Meta-term being either a meta-constant or a meta-variable or meta-function symbol applied to a tuple of meta-terms. e.g. `'f':'father' F'`. etc. When a meta-term expresses a meta-level predicate stating a relation between entities. it has the form `Pred(Sub,Obj)`. and when it

expresses a meta-level predicate stating a characteristic of a property. it has the form **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. We treat the sentence as a Horn-clause **meta-rule**. for the empty tuple in the body we put `true` there instead. e.g.

```
'rdf':'type'('f':'MID','f':'Man') '←' true.

'owl':'propertyDisjointWith'(P,DP) ←
   'owl':'propertyDisjointWith'(DP,P).
```

Meta-statement being a meta-representation of a meta-level sentence accessible by our meta-interpreter. It has two forms *meta_statement(meta-level-sentence)* and *axiom(meta-level-sentence)*. the latter presents a rule for a mathematical axiom. and a built-in atom in SWRL. e.g.

```
meta_statement('rdf':'type'
    ('f':'MID','f':'Man') '←' true).

axiom('owl':'propertyDisjointWith'(P,DP) ←
   'owl':'propertyDisjointWith'(DP,P)).

axiom('swrlb':'builtinAtom'('lessThan',x,y) '←'
    builtin(x < y)).
```

C. Meta-programs of an SWRL Ontology

To formulate meta-programs from SW ontologies to use in our framework. each SWRL ontology is transformed into a meta-program containing a (sub-)meta-program expressed in ML. called MP. and a (sub-)meta-program expressed in MML. called MMP. Another meta-program expresses some mathematical axioms for classes and properties in MML called **AMP** is also needed for the inference engine. i.e. our meta-interpreter. to reason with **MP** and MMP.

- **Meta-program for the object level (MP)**

MP contains information about instances and their relationships in terms of meta-statements for the object level: `statement(P(S,O) ← true)`. and `statement(P(S,O) ← Body)`. where `Body` is either single object-level predicate or conjunction of object-level predicates. The latter form expresses a Horn-clause rule. Here is an example of MP:

```
statement('f':'hasParent'
      ('f':'fa', 'f':'son') ← true).

statement('f':'hasUncle'(X,Z) ←
   'f':'hasParent'(X,Y) ∧ 'f':'hasBrother'(Y,Z)).
```

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

MMP contains meta-statements for classes. properties. their relationships. and class-instance relations in terms of meta-rules. The MMP is represented in the following forms:

`meta_statement(P(S,O) ← true)`,

`meta_statement(P(S,Os) ← true)`, and
`meta_statement(C(Prop) ← true)`,

where `P`. `S`. `O` are predicate. subject. and object of a triple (S. P. O) defined in the ontology. `C` is a characteristic of a property `Prop`. `Os` is a tuple composing of several objects.

Here is a typical example of MMP:

meta-statement about classes and their relationships

```
meta_statement('rdfs':'subClassOf'(C,SC) ← true).
meta_statement('owl':'disjointWith'(C,DC) ← true).
```

meta-statements about properties and their relationships

```
meta_statement('owl':'inverseOf'(P,IP) ← true).
meta_statement('owl':'symmetric'(P) ← true).
```

- **Meta-program for the axioms (AMP)**

AMP contains axioms for classes and properties. they are expressed in the meta-rule forms. In addition, **AMP** also contains axioms for built-in atoms in SWRL. and the purpose of introducing these axioms is to provide a way to translate SWRL built-in atoms in into the corresponding Prolog atoms with matched built-in predicates. Here is a typical sample of **AMP**:

```
axiom('owl':'equivalentClass'(C,EC) ←                        (asec)
      'owl':'equivalentClass'(EC,C)).

axiom('owl':'inverseOf'(P,IP) ←                              (asip)
      'owl':'inverseOf'(IP,P)).

axiom(P(S,O) ←                                               (acip)
      'owl':'inverseOf'(P,IP) ∧ IP(O,S)).

axiom(P(S,O) ← 'owl':'symmetric'(P) ∧ P(O,S)).               (acsmp)

axiom('swrlx':'builtinAtom'('lessThan',x,y) ←                (albia)
      builtin(x < y)).

axiom('swrlx':'builtinAtom'('equal',x,y) ←                   (aebia)
      builtin(x = y)).
```

D. The Meta-interpreter

The meta-interpreter in our framework is used to reason with the meta-programs **MPs**. **MMPs**. and **AMPs** and can be used to develop an intelligent agent to reason with SW ontologies. It is defined by a `demo` predicate of the form `demo(A)`. With this predicate the meta-interpreter can infer an answer `A` from the meta-programs. The interpreter is defined by adapting the Vanilla meta-interpreter [11] for reasoning with the meta-programs. which transformed from SWRL 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. and (3) `axiom(A←B)` for a supporting mathematical axiom. The definition of `demo` is:

```
demo(true).                                          (true)

demo((A) ∧ (B)) ← demo(A) ∧ demo(B).                 (conj)

demo(builtin(BI)) ← BI.                              (btin)

demo(A) ← statement(A ← B) ∧ demo(B).                (ost)

demo(A) ← meta_statement(A ← B) ∧ demo(B).           (mst)

demo(A) ← axiom(A ← B) ∧ demo(B).                    (ast)
```

The first clause (true) is the basic case for proving an atom `true`. The second clause (conj) is used for proving a conjunctive goal. The third clause (btin) is used for

translating SWRL built-in atoms into its corresponding Prolog atoms with matched built-in predicates. The last three clauses (ost), (mst), and (ast) are used for proving three meta statements from the three meta-programs **MP**, **MMP** and **AMP** respectively.

IV. Query Answering with Our Framework

With our framework, SWRL ontologies are transformed into meta-programs **MP**, **MMP**, and **AMP**. The meta-programs are inputs to the meta-interpreter and they are all implemented in Prolog. Then the meta-interpreter is used to derive conclusions from the meta-programs.

We use the family ontology taken from [10] for a demonstration purpose of our meta-interpreter. This is an SWRL ontology, and rules are expressed by the SWRL syntax. Firstly, the ontology is transformed into meta-programs, here we show some parts of them:

- The **MP** program

```
statement('f':'hasParent'                                    (1)
    ('f':'X02','f':'X01') ← true).
statement('f':'hasParent'                                    (2)
    ('f':'X02','f':'F01') ← true).
statement('f':'hasParent'                                    (3)
    ('f': 'X03','f':'X02') ← true .
statement('f':'hasParent'                                    (4)
    ('f': 'X03','f':'X02') ← true).
statement('f':'hasParent'                                    (5)
  ('f':'F02','f':'X03') ← true).
statement('f':'hasAge'('f':'X02',?-) ← true).                (6)
```

//Statements expressing Horn-clause rules

```
statement('f':'hasFather'(?,F) ←                             (r1)
  'f':'hasParent'(?,F)  'rdf':'type'(F,'f':'Man')).
statement('f':'hasMother'(?,M) ←                             (r2)
'f':'hasParent'(?,M)  'rdf':'type'(M,'f':'Woman')).
statement('f':'hasSibling'(P1,P2) ←                          (r3)
  'f':'hasParent'(P1,P3)  'f':'hasParent'(P2,P3)
  'owl':'differentFrom'(P1,P2)).
statement 'f':'hasBrother'(B,S) ←                            (r4)
'f':'hasSibling'(B,S)  'rdf':'type'(B,'f':'Man')).
statement('f':'hasUncle'(P1,P2) ←                            (r5)
'f':'hasParent' P1,P3)  'f':'hasBrother'(P3,P2)).
statement('f':'hasSon'(P,C) ←                                (r6)
'f':'hasChild'(P,C)  'rdf':'type'(C,'f':'Man')).
statement('rdf':'type'(P,'f':'Adult') ←                      (r7)
'f':'hasAge'(P,A)
'swrlx':'builtinAtom' 'lessThan',3,A)).
  ...
```

- The **MMP** program

```
meta_statement('owl':'inverseOf'                             (1')
    ('f':'hasChild','f':'hasParent') ← true ).
meta_statement('rdfs':'subPropertyOf'                        (2')
    f':'hasFather','f':'hasParent') ← true).
meta_statement('rdfs':'subPropertyOf'                        (3')
    f':'hasMother','f':'hasParent') ← true).
meta_statement('owl':'symmetric'                             (4')
    ('f':'hasSibling') ← true).
meta_statement('rdf':'type'                                  (5')
   ('f':'X01','f':'Man') ← true).
meta_statement('rdf':'type'                                  (6')
    ('f':'X02','f':'Man') ← true).
meta_statement('rdf':'type'                                  (7')
    ('f':'X03','f':'Man') ← true).
meta_statement('rdf':'type'                                  (8')
    ('f':'X05','f':'Man') ← true).
meta_statement('rdf':'type'                                  (9')
    'f':'F01','f':'Woman') ← true).
```

We pose some queries to the meta-interpreter and get the answers as the following:

```
?- demo('f':'hasChild'('f':'X01',X)).
  X = 'f':'X02'.
```
The adopted clauses are (ac1p), (ast), (f'), (mst), (1), (ost), and (true).

```
?- demo('f':'hasSon'('f':'F01',X)).
  X = 'f':'X02'.
```
The adopted clauses are (r6), (ost), (conj), (ac1p), (ast), (f'),(mst), (2), (6'), and (true).

```
?-demo('f':'hasFather'('f':'X02',X)).
    X = 'f':'X01'.
```
The adopted clauses are (r1), (ost), (conj), (1), (5'), (mst) and (true).

```
?-demo('f':'hasMother'('f':'X02',X)).
  X = 'f':'F01'.
```
The adopted clauses are (r2), (ost), (conj), (2), (9'), (mst) and (true).

```
?-demo('f':'hasBrother'('f':'X03',X)).
  X = 'f':'X05'.
```
The adopted clauses are (r3), (r4), (ost), (conj) (3), (4), (5'), (mst) and (true).

```
?-demo 'f':'hasUncle'('f':'F02',X)).
  X = 'f':'X05'.
```
The adopted clauses are (r3), (r4), (r5), (ost),(conj), (5), (3), (4), (7'), (mst), (true).

```
?-demo('rdf':'type'(X,'f':'Adult')).
  X = 'f':'X02'.
```
The adopted clauses are (r7), (ost), (conj), (6), (albia), (ast), (bin), and (true).

V. Related Works

The SW research community has addressed similar issues and problems concerning SW ontologies and rules as that also happens in the area of logic programming. So the exchange of idea between these two research areas is inevitable.

For instance, Laera et al. [12] proposed SweetProlog as a system for translating an OWL ontology and rules into a Prolog program. It is achieved by the translation of an OWL ontology described in Description Logic and rules expressed in OWLRuleML into a set of facts and a set of rules in Prolog respectively. Then any reasoning on these facts and rules can be performed by the Prolog interpreter.

Comparing this with our work, according to their approach an OWL and RuleML ontology is entirely translated into Prolog facts and rules, where both object level and meta level knowledge of the ontology are mixed up and Laera et al. do not care to make the distinction between the two levels of knowledge, whilst our translation makes a careful separation between the two levels of knowledge. As a result, their SweetProlog can reason with any object level knowledge of an ontology as the way our approach does.

For example, with the same ontology that we use for the query answering in Section 4, according to their approach, the ontology would be transformed to Prolog facts as follows:

```
hasParent('f':'F02','f': 'X01').
hasBrother('f': 'X03', 'f': 'X05').
...
```

and the 'uncle' rule would be expressed by the Prolog rule:

```
hasUncle(X,Y):- hasParent(X,Z), hasBrother(Z,Y).
```

Provided with this prolog program and a query like

```
?-hasUncle('F1', X).
```

their SweetProlog would give the answer `X = 'F5'; X = 'F6'.` which is the same answer as that given earlier by our meta-interpreter.

However, there could be queries, which ask about meta level information of this ontology, which their SweetProlog cannot give answers to, since there is some information that can be asked only at the meta level, but cannot do that at the object level.

For example, a query asking what the relation between `M12` and `M13` is:

```
?-P('M2','F3','M13').
```

So, in SweetProlog, the Prolog interpreter will signal *a syntax error*, since a variable is not allowed to be used as a predicate name in a query. Here a predicate name is a meta level information that cannot be asked at the object level.

However, with a careful treatment of a separation between the object level and the meta level knowledge in our approach, such a query can be asked via the `ask` predicate as follows.

```
?-ask(P('M2','M13')).
```

and our meta-interpreter can give the answer:

```
P = hasUncle.
```

In the same direction as Laera et al., Samuel et al. proposed SWORIER [13], which also translates an SWRL ontology into Prolog facts and rules, and derives answers from the Prolog program using the Prolog interpreter. SWORIER suffers the same problem as SweetProlog does due to the similar approach of making no distinction between the object and meta levels. Comparing our work with theirs, in SWORIER Samuel et al. defined a set of *General Rules* in Prolog in order to formulate the OWL primitives. Here their *General Rules* serve the same purpose as our AMP program.

Another related work is a work on query answering for OWL-DL with rules [14]. In this work, OWL-DL was extended with DL-safe rules in order to provide deduction on an OWL-DL ontology. An undecidability problem of deduction on OWL-DL ontology with rules is solved by making some restrictions in DL-safe rules. This approach provides query answering algorithm that can handle only partial OWL-DL, since some axioms of OWL-DL, such as the transitivity axioms, are taken out from OWL-DL in order to maintain the decidability of their query answering algorithm. This work proposed a deduction method by means of a specific algorithm whereas we adopt a general purpose inference engine based on metalogic.

VI. Conclusion

We have presented a meta-logical framework for reasoning with an SWRL ontology. In this paper, our previous framework that was designed to support OWL has been extended to accommodate SWRL rules by improving the meta-languages to express Horn-clause rules and modifying the meta-interpreter so that it can work with the newly revised meta-languages.

Acknowledgment

We gracefully acknowledge the financial support for this research from the Japan International Corporation Agency (JICA) under the AUN/SEED-Net Project.

References

[1] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P. F. Patel Schneider Eds.). *The Description Logic Hand book: Theory, Implementation, and Applications*. 2nd ed. Cambridge, 2007.

[2] W3C. The Resource Description Framework. http: www.w3.org/RDF/.

[3] P. F. Patel-Schneider, P. Hayes, I. Horrocks. OWL Web Ontology Language Semantics and Abstract Syntax. W3C Recommendation. http: www.w3.org tr 2004 rec-owl-semantics-20040210/.

[4] Boris Motik, Peter F. Patel-Schneider, and Bernardo Cuenca Grau. *SWRL: A semantic web rule language combining OWL and RuleML*. URL, 2009. http: www.w3.org Submission SWRL .

[5] V. Hirankitti, and V. N. Tran. *A Meta logical Approach for Reasoning with Semantic Web Ontologies*. In proc. of the 4th IEEE Int. Conf. on Computer Sciences: Research, Innovation & Vision for the Future, 2006.

[6] V. Hirankitti, and V. N. Tran. *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, Lecture Notes in Computer Science: Vol. 4369, Springer-Verlag, pp. 215-228, 2006.

[7] V. Hirankitti, and M. N. Trang. *A Meta-logical Approach for Reasoning with an OWL 2 Ontologies*. In proc. of the 2010 IEEE Int. Conf. on Computer Sciences: Research, Innovation & Vision for the Future, 2010.

[8] Rule Markup Language, URL, http: www.ruleml.org .

[9] Masahiro Hori, Jerome Euzenat, and P. F. Pate-Schneider. *OWL Web Ontology Language XML Presentation Syntax*. URL, 2003. http: www.w3.org TR owl-xmlsyntax .

[10] The family ontology, URL, http: protege.cim3.net file pub ontologies family family.swrl.owl 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.

[12] L. Laera, V. A. M. Tamma, and G. Semeraro. *SweetProlog: A System to Integrate Ontologies and Rules*. In Proc. Of RuleML'04, pp. 188-193, Springer Verlag, 2004.

[13] K. Samuel, L. Obrst, S. Stoutenberg, K. Fox, P. Franklin, A. Johnson, K. Laskey, D. Nichols, S. Lopez, and J. Peterson. *Translating OWL and Semantic Web rules into Prolog: Moving toward description logic programs*. Journal Theory and Practice of Logic Programming, pp. 301-322, 2008

[14] B. Motik, U. Sattler, and R. Studer *Query Answering for OWL-DL with Rules*. In Proc. ISWC2004, pp. 549-563, Springer, November 2004.

A Meta-logical Approach for Reasoning with Ontologies and Rules in OWL 2

Visit Hirankitti, and Trang Mai Xuan

Abstract— **OWL became a de facto language for representing Semantic Web ontologies. Having been improved and extended with rules, the language was later updated to OWL 2. In this paper we propose a framework for reasoning with an OWL2 ontology and rules using meta-logic. Our meta-logical system consists of meta-programs expressing ontologies and rules at the meta-level, and an inference engine in a form of meta-interpreters defined by a demo(.) predicate. The framework reasons with ontologies and rules in OWL 2 by first an ontology and rules being translated into meta-statements, and these meta-statements then being reasoned by the meta-interpreter, which provides a query answering mechanism to infer implicit information. A comparative study of related works has revealed a merit of the meta-logical representation approach that separates the meta level knowledge from the object level one.**

Index Terms— **Semantic Web Ontologies, OWL 2, Rules, Metalogic, Meta-reasoning.**

I. Introduction

Ontologies and rules have played an important role in the Semantic Web (or shortly 'SW'). An ontology forms vocabularies and sentences used to express knowledge, and this knowledge can be shared on the web. OWL was accepted by W3C as a language for representing a web ontology. Its core, OWL-DL, is essentially an XML encoding of an expressive Description Logic (DL) built upon RDF (Resource Description Framework) with a substantial fragment of RDF-Schema (RDFS). The vocabularies defined in such an ontology consist of classes (or so-called concepts) and properties (or so-called roles); in logic classes can be treated as unary predicates while properties as binary predicates, and all these predicates represent relations. OWL was successfully applied to the semantic web in the past. However, some knowledge should be formulated more naturally as rules rather than axioms.

A rule representation is the formalism used in logic programming. It has been proposed as a promising approach for representing knowledge in SW which complements to other means of knowledge representation in OWL. In the broadest sense, a rule can be any statement of the form "if the precondition p holds then the conclusion c holds", where the precondition and the conclusion are logical sentences. The realization of rules allows a means to deduce and combine information. This leads to a way for enhancing content, and supporting reasoning capabilities, on OWL ontologies.

The extension of SW ontologies with rules has recently attracted much attention in the Semantic Web research, and many approaches have been proposed for it. One of them is to combine DL with first-order Horn-clause rules. This is the basis of the Semantic Web Rule Language, SWRL [2], a language for rule formulation and rule extension to OWL. However, inferences on SWRL rules can lead to undecidability even though the rules are assumed to be function-free [2]. In order to make the inferences decidable, some restrictions were put upon the rule language in the form of DL-safe rules [7] or of Description Logic Programs DLP [1]. Recently, a new revision of OWL has been developed by W3C, it is called "OWL 2". OWL 2 has much improvement on its predecessor—OWL—especially with rule formulation based on DL *SROIQ* [11], in which DL rules can be completely ensured as decidable fragment of SWRL.

In our previous work [9], we have developed a meta-logical approach for reasoning with semantic web ontologies expressed in OWL. In this paper we go further by extending that framework so that it can reason with SW ontologies and rules in OWL 2.

The remainder of the paper is organized as follows. Section II reviews some concepts of ontologies with a rule extension, and shows how rules can be expressed in OWL 2. Accordingly, we extend our previous work in order to reason with ontologies and rules expressed in OWL 2 in Section III. Section IV demonstrates how our framework reasons with ontolologies and rules. We discuss related work in section V. Finally, section VI concludes this work.

II. Extending Ontologies with Rules

A. SW Ontologies with Rules

Adding rules to ontologies expressed in OWL could be regarded as an important step forward in the SW research, as inferences can now be made upon SW ontologies; and many research proposals have been proposed; they range from hybrid approaches to homogeneous ones.

The idea behind the former approaches was that the predicates in the rules and predicates in the ontologies are made distinguished, and suitable interface between them is provided. The research works on this direction are such as $\mathcal{AL}$-log language, a hybrid integration of Datalog and

Visit Hirankitti is with the School of Computer Engineering, Faculty of Engineering, King Mongkut's Institute of Technology Ladkrabang, Ladkrabang Dist., Bangkok 10520, Thailand (e-mail: v_hirankitti@yahoo.com).

Trang Xuan Mai is with the International College, King Mongkut's Institute of Technology Ladkrabang, Bangkok 10520, Thailand (e-mail: trangxm@gmail.com).

Description Logic *ALC* [4], CARIN, another hybrid integration of Datalog with different DLs [5], and a hybrid integration of OWL DL (or more precisely the DL *SHOIN*) with normal rules under answer set semantics [6].

According to the latter approaches both rules and ontologies are combined into the same logical language without making a priori distinction between the rule predicates and the ontology predicates. Two example languages based on these approaches are Description Logic Program (DLP) [1] and Semantic Web Rule Language (SWRL) [2]. In [1] a DLP was proposed by combining Description Logics (DLs), which is the basis for the ontology languages, with Logic Programs (LP), which is the basis for rule languages. It supports a bidirectional translation of premises and inferences between the fragment of DL and LP, and vice versa. This translation enables one to construct rules on top of SW ontologies. Later SWRL was proposed in [2] as a new language for an integration of rules and ontologies, in which OWL was extended with Horn-clause rules expressed in RuleML.

Recently SWRL becomes the most widely used language for describing ontologies with rules. However, the straightforward addition of the rules to ontologies leads to undecidability when reasoning with SWRL ontologies. In order to retain decidability, some restrictions have been put upon SWRL rules, and these restricted rules can be rewritten as a set of DL axioms using features introduced in *SROIQ*. This technique has been presented in [8]. Due to the fact that OWL 2 was developed based on DL *SROIQ*, OWL 2 can therefore express rules in the form of DL axioms.

B. Expressing Rules in OWL 2

Some previous SW ontology languages such as DAML+OIL and OWL were developed based on DL *SHOIQ* [12]. *SHOIQ* provides a variety of constructors for building class expressions. The DL class expressions can be demonstrated as being corresponded to first order logic (FOL) which is used to formulate rules. Table I shows the correspondence between FOL formulae and the DL class expressions [1].

According to this correspondence, an FOL sentence can be expressed in DL as well as in DAML+OIL or OWL. For example, a rule of the form: $C(x) \wedge \neg D(x) \rightarrow E(x) \vee F(x)$ can be rewritten as a DL axiom: $C \sqcap \neg D \sqsubseteq E \sqcup F$, and a rule of the form: $C(x) \wedge R(x,y) \rightarrow E(x)$ can be rewritten in DL as the axiom: $C \sqcap R.\top \sqsubseteq E$. However, with some rules such as:

$$hasParent(x,y) \wedge hasBrother(y,z) \rightarrow hasUncle(x,z), \text{ and} \quad (A)$$

$$Man(x) \wedge hasChild(x,y) \rightarrow fatherOf(x,y). \quad (B)$$

There is no correspondence so they cannot be rewritten as DL axioms, however to be able to do so we need some extra axioms in DL *SROIQ*, these are some new features introduced in OWL 2.

OWL 2 which is based on DL *SROIQ*[11] supports the Role Inclusion Axiom (RIA) or so-called the "property chain" axiom and the Self concept, and these can be used to express more forms of rules, including the previous example.

Table I: DL-FOL equivalence

Expression	DL	FOL
subclassOf	$C \sqsubseteq D$	$\forall x.C(x) \rightarrow D(x)$
subpropertyOf	$P_1 \sqsubseteq P_2$	$\forall x,y.P_1(x,y) \rightarrow P_2(x,y)$
transitiveProperty	$P^+ \sqsubseteq P$	$\forall x,y,z.(P(x,y) \wedge P(y,z)) \rightarrow P(x,z)$
functionalPropery	$\top \sqsubseteq \leq 1\ P$	$\forall x,y,z.(P(x,y) \wedge P(x,z)) \rightarrow y=z$
inversePropery	$P \equiv Q^-$	$\forall x,y.P(x,y) \rightarrow Q(y,x)$
intersectionOf	$C_1 \sqcap \ldots \sqcap C_n$	$C_1(x) \wedge \ldots \wedge C_n(x)$
unionOf	$C_1 \sqcup \ldots \sqcup C_n$	$C_1(x) \vee \ldots \vee C_n(x)$
complementOf	$\neg C$	$\neg C(x)$

The Role Inclusion Axioms are the constructs of the form $R \circ S \sqsubseteq T$ where $\circ$ is a binary composition operator. This form is equivalent to an FOL formula: $\forall x,y,z.(R(x,y) \wedge S(y,z)) \rightarrow T(x,z)$. By adopting this, rule A can easily be rewritten as a DL *SROIQ*axiom:

$$hasParent \circ hasBrother \sqsubseteq hasUncle.$$

The Self concept allows one to express a "local reflexive" property, e.g. $R(x,x)$, in which a role R relates an individual x to itself. The Self concept can be used to transform a property $R(x,x)$ into a class C_s and vice versa, and this is due to a DL *SROIQ*axiom $C_s \equiv \exists R.Self$.

Therefore to derive the DL *SROIQ*axioms corresponding to rule B, we first transform a class Man into a property P_{Man} by introducing an axiom $Man \equiv \exists P_{Man}.Self$, we then apply the previous RIA. As a result, rule B will be equivalent to the DL *SROIQ*axioms:

$$Man \equiv \exists P_{Man}.Self.$$
$$P_{Man} \circ hasChild \sqsubseteq fatherOf.$$

With the DL-FOL and DL *SROIQ*-FL mappings we can express a rather wide range form of rules in DL axioms of OWL 2 and vice versa, if they satisfy certain restrictions [8]. In the next section we extend our previous framework [9] so that it can reason with ontologies and rules in OWL 2.

III. Our Meta-logical Approach

A. Our Approach

Our framework forms a logical system consisting of meta-programs 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 described by OWL 2 is transformed into a meta-logical representation. The later is a meta-interpreter, in the form of a demo (meta-)program, which is used to infer explicit as well as implicit information, or in other words draw conclusions, from the former. The meta-interpreter can also 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.

In this paper our previous framework [9], which was designed to support reasoning with OWL ontologies, is now

enhanced with an ability to reason with ontologies and rules expressed in OWL 2. Our meta-logical system can simply be illustrated in Fig. 1. In order to support rules, which are expressed by using the new features in OWL 2, the meta-program in our previous framework has to be extended with some new forms of meta-statements.

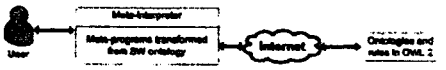


Fig. 1 Our meta-logical system

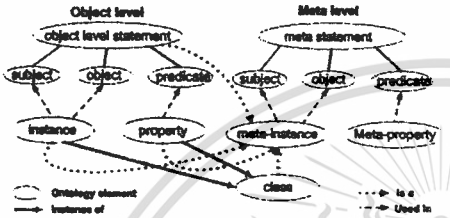


Fig. 2 Object level and Meta level of ontology elements.

To explain our framework, in the next three sub-sections we first introduce our meta-language used for formulating the meta-programs of ontologies and rules, then explain the meta-programs in details. Finally we describe our meta-interpreter.

B. Meta-language for an OWL2 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 depicted in Fig. 2. At the object level, an instance can be an individual or a literal of a domain, e.g. 'john', and property is a relationship between individuals, or is an individual's attribute, e.g. 'hasSon', 'type'. At the meta-level, a meta-instance can be an individual, a property, a class, or an object-level statement. A meta-property is a property to describe a meta-instance's attribute or a relationship between/among meta-instances, e.g. 'reflexive', etc. 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>, such as 'f':'son', 'f':'hasSon', 'owl':'reflexive', etc. Henceforth this qualified name will be used throughout.

According to our framework, in an SW ontology we distinguish between its object and meta levels, and similarly its object and meta languages. The object language specifies objects and their relationships in the real world. The meta-language describes the syntactic form of the object language. Hence, we have formulated two meta-languages: one discussing mainly about objects and their relationships we call it "meta-language for the object level (ML)", and the other we call "meta-language for the meta-level (MML)," which discusses mainly about classes, instances, properties and their relationships.

- **Meta-language for the object level (ML)**

Objects and their relationships at the object level are specified in an SW ontology and this information is expressed by the elements of ML below.

Meta-constant specifies a name of an object and a literal, e.g. `'son'`, including a reference, e.g. a namespace, the latter is a meta-constant of MML. This means that ML and MML are not totally 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 objects, or a name of an object's property—i.e. an object-level predicate name, such as `'hasSon'`, `'name'` It also stands for other meta-level function symbol, e.g. `'←'`, `'∧'`, `':'`.

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

```
'f':'hasSon'('f':'fa','f':'son').
```

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':'hasSon'('f':'fa','f':'son')←true).
```

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

Apart from the object language, an SW ontology also defines classes, properties, their relationships, as well as class-instance relations, and we argue that this information is *meta-information of the object level*. Here we express this information by MML which 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. `'←'`, `'∧'`, `'¬'` (`'¬'` is used to express a classical negation); or `':'`; or a name of set operators applied on classes such as union; or a meta-predicate name being a name of a relation between entities; or a name of characteristic of a property, which may fall into one of the following categories:

Class-class relations: equivalent class of, etc.
Class-instance relations: instance of, class of, etc.
Property-property relations: property chain of, etc.
Class-property relations: Keys, etc.
Relations between literals and instances/classes/properties: we can take these relations as attributes of instances, of classes, or of properties, e.g. comment.
Characteristics of properties: reflexive, asymmetric, etc.

Meta-term being either a meta-constant or a meta-variable or meta-function symbol applied to a tuple of meta-terms, e.g. `'f':'fatherOf'`, etc. 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(Pro)`, where `Pred` is a meta-predicate name, `Sub`, `Obj`, and `Pro` (a property) are meta-constants or meta-variables.

An example of a meta-term expressing a classical negation, is in the form of `'¬''rdf':'type'(I,C)`, where `I` is a meta-constant specifying an individual, and `C` is a meta-constant specifying a class.

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':'propertyDisjointWith'(P,DP) ←
   'owl':'propertyDisjointWith'(DP,P).
```

Meta-statement being a meta-predicate or meta-predicates connected by logical connective. It has two forms meta_statement(meta-level-sentence) and axiom(meta-level-sentence), the latter represents a rule for a mathematical axiom, e.g.:

```
meta_statement('owl':'propertyDisjointWith'
      ('f':'likes','f':'dislikes') ← true).
axiom('owl':'propertyDisjointWith'(P,DP) ←
    'owl':'propertyDisjointWith'(DP,P)).
```

C. *Meta-programs of an Ontology with Rules*

Each OWL2 ontology is transformed into a meta-program containing a (sub-)meta-program expressed in **ML**, called **MP**, and a (sub-)meta-program expressed in **MML**, called **MMP**. Another meta-program expresses some mathematical axioms for classes and properties called **AMP** is also needed for the inference engine to reason with **MP** and **MMP**.

- **Meta-program for the object level (MP)**

MP contains information of instances and their relationship in the form of meta-statements for the object level: `statement(P(S,O) ← true)`. In terms of the rule system, it can be understood as "facts". Here is an example:

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

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

MMP contains meta-statements for classes, properties, their relationships, and class-instance relationships in the form of meta-rules. Here are some typical examples:

Some meta-statement about classes and their relationships:

```
meta_statement('rdfs':'subClassOf'
  (C,SC) ← true).
// The class C is sub-class of class SC.

meta_statement('rdfs':'equivalentClass'
  (C,EC) ← true).
// Classes C and EC are equivalent.

meta_statement('owl':'disjoinwith'
  (C,DC) ← true).
// Classes C and DC are disjoint.

meta_statement('owl':'intersectionOf'
  (C,Cs) ← true).
// Class C is intersection of classes in Cs.

meta_statement('owl':'unionOf'
  (C,Cs) ← true).
// Class C is union of classes in Cs.

meta_statement('owl':'complementOf'
  (C,CC) ← true).
// Class C is complement of class in CC.

meta_statement('rdf':'type'(I,C) ← true).
// The Instance I is an instance of class C.
...
```

Meta-statements about properties and their relationships:

```
meta_statement('owl':'inverseOf'
  (P,IP) ← true).
//The property P is an inversion of property IP.

meta_statement('owl':'symmtric'(P) ← true).
//The property P is symmetric.

meta_statement('rdfs':'domain'(P,D) ← true).
//The domain of property P is D.
...
```

The new features in OWL 2 that referred to in section II can be translated to the following meta-statements in **MMP**:

```
meta_statement('owl':'propertyChainOf'
  (P,[P1, P2]) ← true).
//Express RIA: Property P is composition of properties P1, P2.

meta_statement('owl':'objectHasSelf'(
  C, Pc) ← true).
//Express the Self concept: C is a class of individuals which are
related to themselves under role Pc.
```

With such meta-statements we can transform DL rules into a meta-program. Here are examples of **MMP** that correspond to the rules we listed in section II.B:

Rule "$C \sqcap \neg D \sqsubseteq E \sqcup F$" transforms to MMP:

```
meta_statement('rdfs':'subClassOf'
  (M,N) ← true).
meta_statement('rdfs':'unionOf'
  (N,[E,F]) ← true).
meta_statement('rdfs':'intersectionOf'
  (M,[C,D']) ← true).
meta_statement('rdfs':'complementOf'
  (D',D) ← true).
```

Rule "hasParent ∘ hasBrother ⊑ hasUncle" transforms to MMP:

```
meta_statement('owl':'propertyChainOf'
  ('f':'hasUncle',['f':'hasParent',
  'f':'hasBrother']) ← true).
```

Rule "`Man(x) ∧ hasChil (x,y) → fatherOf(x,y)`" transforms to **MMP**:

```
meta_statement('owl':'objectHasSelf'
  ('f': 'Man', PMan) ← true).
meta_statement('owl':'propertyChainOf'
  ('f':'fatherOf',
  [PMan,'f':'hasChild']) ← true).
```

- **Meta-program for the axioms (AMP)**

AMP contains axioms for classes and properties, they are expressed in the meta-rule form. In [9], we had several axioms in **AMP** to support for OWL. In order to work with ontologies and rules expressed in OWL 2, we *add more axioms* to manipulate with the new features in OWL 2, and an axiom for a set complement. Here we list all the axioms corresponding to the formulae in Table I and to the new features in OWL 2:

```
axiom('rdf':'type'(I,C) ←                          (asic)
    'owl':'subclassOf'(SC,C) ∧
    'rdf':'type'(I,SC)).
// Axiom to handle subclass formulae.

axiom(P(S,O) ←                                     (acsp)
  'rdfs':'subPropertyOf'(SP,P) ∧ SP(S,O)).
// Axiom to handle subproperty formulae.
```

```
axiom(P(S,O) ←                                              (actp)
  'owl':'transitive'(P) ∧ P(S,O1) ∧ P(O1,O)).
// Axiom to handle the transitive property

axiom(P(S,O) ←                                              (acfp)
  'owl':'functional'(P) ∧
  P(S,O1) ∧ 'owl':'sameAs'(O,O1)).
// Axiom to handle the functional property

axiom(P(S,O) ←                                              (acip)
  'owl':'inverseOf'(P,IP) ∧ IP(O,S)).
// Property IP is an inverse property of P.

axiom(P(S,O) ←                                              (acpc)
  'owl':'propertyChainOf'(P,[P1,P2]) ∧
   P1(S,O1) ∧ P2(O1,O)).
// Axiom to handle the chain property.

axiom(P(S,S) ←                                              (acsc)
  'owl':'objectHasSelf'(C,P) ∧
  'rdf':'type'(S,C)).
// Axiom to handle the Self concept.

axiom('rdf':'type'(I,C) ←                                   (acic)
  'owl':'intersectionOf'(C,Cs) ∧
   'intertype'(I,Cs)).
'intertype'(I,[H|T]) ←
  'rdf':'type'(I,H) ∧ 'intertype'(I,T).
// Axiom to handle a set intersection.

axiom('rdf':'type'(I,C) ←                                   (acuc)
  'owl':'unionOf'(C,Cs) ∧ 'unionType'(I,Cs)).
'unionType'(I,[H|T]) ← 'rdf':'type'(I,H).
'unionType'(I,[H|T]) ← 'unionType'(I,T).
// Axiom to handle a set union.

axiom('¬' 'rdf':'type'(I,C) ←                               (accc)
   'owl':'complementOf'(C, Cc) ∧
   'rdf':'type'(I,Cc)).
// Axiom to handle a set complement.
```

D. The Meta-interpreter

The meta-interpreter in our framework is constructed for reasoning with the meta-programs **MPs**, **MMPs**, and **AMPs** and can be used to develop an intelligent agent to reason with SW ontologies. It is defined by a `demo` predicate of the form `demo(A)`. With this predicate we can infer the answer `A` from the meta-programs. Our meta-interpreter adapts the Vanilla meta-interpreter in [10] in order for reasoning with the meta-programs transformed from ontologies and rules where we have defined 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 (including rules), and (3) `axiom(A ← B)` for the mathematical axioms. The definition of `demo/1` is:

```
demo(true).                                                 (true)

demo(A' ∧ 'B) ← demo(A) ∧ demo(B).                          (conj)

demo(A) ← statement(A' ← 'B) ∧ demo(B).                     (ost)

demo(A) ← meta_statement(A' ← 'B) ∧ demo(B).                (mst)

demo(A) ← axiom(A' ← 'B) ∧ demo(B).                         (ast)
```

The first clause (true) is the basic case for proving that an atom is true. The second clause (conj) is used for proving a conjunction goal. Three last clauses (ost), (mst), and (ast) are used for interpreting three meta statements of the three meta-programs **MP**, **MMP**, and **AMP** respectively.

IV. Query Answering with Our Framework

In our framework ontologies and rules expressed in OWL 2 are transformed into meta-programs. The meta-programs are formed by three sub meta-programs **MP**, **MMP** and **AMP**. The meta-programs are used as inputs of the meta-interpreter which is implemented in Prolog. The meta-interpreter is an inference engine reasoning with meta-programs to derive conclusions from the ontologies with rules.

The family ontology [13] is used as an example to demonstrate our framework. Due to its lack of rules, three OWL2 rules are added to it. After the whole ontology is transformed into meta-programs, here are some parts of them:

- The **MP** program

```
statement('f':'hasParent'                                   (1)
  ('f':'M02','f':'M01') ← true).

statement('f':'hasParent'                                   (2)
  ('f':'F02','f':'M01') ← true).

statement('f':'hasParent'                                   (3)
  ('f':'M03','f':'F02') ← true).
...
```

- The **MMP** program

```
meta_statement('rdf':'type'                                 (1')
  ('f':'M01','f':'Man') ← true).

meta_statement('rdf':'type'                                 (2')
  ('f':'M02','f':'Man') ← true).

meta_statement('rdf':'type'                                 (3')
  ('f':'F02','f':'WoMan') ← true).

meta_statement('owl':'complementOf'                         (4')
  ('f':'Man','f':'WoMan') ← true).

meta_statement('owl':'unionOf'                              (5')
  ('f':'Human',
   ['f':'Man','f':'WoMan']) ← true).
```

The first rule `hasParent(x,y) ∧ hasParent(z,y) → siblingOf(x,z)` added is expressed by `hasParent ∘ hasParent⁻ ⊑ siblingOf` DL axiom, where `hasParent⁻` is the inverse property of `hasParent`. This is transformed into the following meta-statements in **MMP**:

```
meta_statement('owl':'inverseOf'                            (6')
  ('f':'parentOf','f':'hasParent') ← true).

meta_statement('owl':'propertyChainOf'                      (7')
  :'f':'siblingOf','f':'hasParent',
  'f':'parentOf']) ← true ).
```

The next one `Man(x) ∧ siblingOf(x,y) → brotherOf(x,y)` is transformed into the meta-statements in **MMP**:

```
meta_statement('owl':'objectHasSelf'                        (8')
  :'f': 'Man', P_Man) ← true).

meta_statement('owl':'propertyChainOf'                      (9')
  ('f':'brotherOf',
  [P_Man,'f':'siblingOf']) ← true).
```

The third rule `brotherOf(x,y) ∧ parentOf(y,z) → uncleOf(x,z)` is transformed into the following meta-statements in **MMP**:

```
meta_statement('owl':'propertyChainOf'                      (10')
   ('f':'uncleOf',['f':'brotherOf',
   'f':'parentOf']) ← true ).
```

Now we pose some queries to the meta-interpreter to get

answers as follows:

```
?- demo('rdf':'type'('f':'M02',X)).          (q1)
  X = 'f':'Man';
  X = 'f':'Human'.
```

//1st answer is supported by (mst), (1'), and (true), and 2nd answer is supported by (ast), (acuc), (conj), (5'), (1'), and (true).

```
?- demo('¬''rdf':'type'('f':'F02',X)).       (q2)
  X = 'f':'Man'.
```

//The adopted clauses are (ast), (accc), (conj), (mst), (4'), (3') and (true).

```
?- demo('f':'siblingOf'('f':'M02',X)).       (q3)
  X = 'f':'F02'.
```

//The adopted clauses are (ast), (acpc), (conj), (7'), (true), (ost), (1), (mst), (acip), (6'), (2).

```
?- demo('f':'brotherOf'(X, 'f':'F02')),      (q4)
  X = 'f':'M02'.
```

//The adopted clauses are (ast), (acpc), (conj), (9'), (true), (acsc), (8'), (mst), (2'), and the clauses adopted for answering q3.

```
?- demo('f':'uncleOf'('f':'M02',X)).         (q5)
  X = 'f':'M03'.
```

//The adopted clauses are (ast), (acpc), (conj), (10'), (true), (3), and the clauses adopted for answering q4.

V. Related Works

We now look at other approach that enhances ontologies with rules. Grosof et al. [1] proposed the Description Logic Program (DLP); this approach supports bi-directional translation between logical sentences from DLP fragment of Description Logic and logic programs.

According to this approach, every concept referred to in an ontology is mapped into a unary relation with a concept name becoming a name of the relation and an individual name becoming an argument. Every instance-property-instance relationship is mapped into a binary relation. In addition, concepts as well as property constructor statements are converted into rules. The distinction between this approach and ours is the following.

Firstly this approach was designed to support a subset of DAML+OIL, and provides only a mapping from RDFS and DAML+OIL to logic programs, but does not support the new features in OWL 2, such as the property chain axiom and the Self concept. Secondly, this approach has a weakness when representing an ontology in a logic program. For example, to represent the statement "a is union of b_1, b_2, ..., b_n", it requires n number of rules to do so, i.e. `a(X):-b1(X)`, ... `a(X):-bn(X)`. However, in our representation, this requires only one statement, that is our (acuc) axiom, which is more compact.

Even more importantly, their representation of logic program is at the object level only. Thus, the names of concepts, or the names of roles, of an ontology which are meta-terms cannot be accessed and reasoned from their logic program and therefore cannot be queried by an inference engine. For example, in their representation, statement (1) and (1') in section IV would be presented as the following facts:

```
hasParent('f':'M02','f':'M01').
Man('F':'M01').
```

With these clauses we can ask only question who is a man or who is a parent of `'M02'`, but it is impossible to get answers to a question like which class `'M01'` is an instance of, or what a relationship between `'M02'` and `'M01'` is. This is because a predicate name is a meta-level information that cannot be reasoned and queried at object-level by a Prolog interpreter. However, in our approach since we separate the meta-level from the object-level knowledge in an ontology, such queries can be asked and answered via `demo` predicate:

```
?-demo(P('f':'M02','f':'M01')).
  P='f':'hasParent'.

?-demo('rdf':'type'('f':'M01','f':X)).
X=Man.
```

VI. Conclusion

In this paper we have presented a meta-logical framework for representing and reasoning with ontologies and rules expressed in OWL 2. The logical system of our framework consists of meta-programs transformed from ontologies and rules expressed in OWL 2, and an inference engine defined by a demo predicate with the new extra auxiliary axioms proposed in the paper.

Acknowledgment

We gracefully acknowledge the financial support for this research from the Japan International Corporation Agency (JICA) under the AUN/SEED-Net Program.

References

[1] Benjamin N. Grosof, I. Horrocks, Raphael Volz, Stefan Decker, *Description Logic Programs: Combining Logic Programs with Description Logic*, In Proc. of the 12th Int'l Conf. on the WWW, pp. 48-57, ACM, 2003.

[2] Ian Horrocks, Peter F. Patel-Schneider, *A Proposal for an OWL Rules Language*, In Proc. of the 13th Int'l Conf. on the WWW, ACM, 2004.

[3] Jakob Henriksson, Jan Maluszyński, *Hybrid integration of rules and ontologies: A constraint-based framework*, Rule and ontology integration workshop, RuleML 2006 Athens, GA, USA, 2006.

[4] Francesco M. Donini, Maurizio Lenzerini, and Andrea Schaerf, *AL-log: Integrating datalog and description logics*, Journal of Intelligent Information Systems, p. 227 – 252, 1998.

[5] Alon . Levy and Marie-Christine Rousset, *Combining Horn rules and description logics in CARIN*, Artificial Intelligence, pp. 165 – 209, 1998.

[6] T. Eiter, T. Lukasiewicz, R. Schindlauer, and H. Tompits, *Combining Answer Set Programming with Description Logics for the Semantic Web*, In Proc. Ninth International Conference of Principles of Knowledge Representation and Reasoning (KR2004), p. 141 – 151, 2004.

[7] Boris Motik, Ulrike Sattler, and Rudi Studer, *Query answering for OWL-DL with rules*, Journal of Web Semantics: Science, Services and Agents on the World Wide Web, 3(1)41-60, 2005.

[8] F. Gasse, U. Sattler, and V. Haarslev, *Rewriting Rules into SROIQ Axioms*, Poster at 21st Int. Workshop on DLs (DL-08), 2008.

[9] V. Hirankitti, and V. X. Tran, *A Meta logical Approach for Reasoning with Semantic Web Ontologies*, In proc. of the 4th IEEE Int. Conf. on Computer Sciences: Research, Innovation & Vision for the Future, Vietnam, pp. 228-235, 2006.

[10] 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.

[11] I. Horrocks, O. Kutz, U. Sattler, *The even more irresistible SROIQ*, In Proc. of the 10th Int. Conf. On Principles of Knowledge Representation and Reasoning, 2006, pp. 57-67, AAAI Press. I.

[12] P. F. Patel-Schneider, P. Hayes, and I. Horrocks, *OWL Web Ontology Language: Semantics and Abstract Syntax*, W3C Recommendation, http://www.w3.org/TR/owl-semantics/, Feb. 2004.

[13] The family ontology, http://www.owldl.com/ontologies/family.owl.