

國立交通大學

資訊管理研究所

博士論文

一個擴充物件導向設計模式以存取語意網類別的方法

A method for extending object oriented programming to access
semantic web classes

研究生：邱泊寰

指導教授：羅濟群博士, 趙國銘博士

中華民國九十九年一月

一個擴充物件導向設計模式以存取語意網類別的方法
**A method for extending object oriented programming to
access semantic web classes**

研究生：邱泊寰
指導教授：羅濟群, 趙國銘

Student : Po-Huan Chiu
Advisor: Chi-Chun Lo, Kuo-Ming Chao



Submitted to Institute of Information Management
College of Management
National Chiao Tung University
in Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy in Information Management
January 2010
Hsinchu, Taiwan, the Republic of China

中華民國九十九年一月

一個擴充物件導向設計模式以存取語意網類別的方法

研究生：邱泊寰

指導教授：羅濟群博士, 趙國銘博士

國立交通大學資訊管理研究所

摘要

物件導向設計已經成為目前軟體開發工具的主流方向。然而物件導向設計的概念中卻缺乏了語意網的概念，導致異質資料交換整合時，無法很自然的採用物件的表示法進行程式開發與類別操作。有許多需要突破的項目如下：使用物件方法來操作RDF資料、全自動將物件資料轉換為RDF格式、以及讓各種物件導向語言都支援相同的語意網開發方法。另外，物件導向類別在定義時有很大的限制，目前仍無法自然的表達出與RDF格式相似的語意關連性，語意網類別和物件導向類別存在著本質上的差異而無法直接對映。我們希望藉由提出一個新的開發方法和概念：語意物件框架，以結合物件導向設計和語意網兩者的優點。語意物件框架使用嵌入式註解來描述物件導向類別和屬性之間的語意關係。我們將採用一個行動電話設計案例，以展示語意物件框架的開發方法。

關鍵字：物件導向設計, 語意網, 設計模式, 資源描述結構

A method for extending object oriented programming to access semantic web classes

Student: Po-Huan Chiu

Advisor: Chi-Chun Lo, Kuo-Ming Chao

Institute of Information Management
National Chiao Tung University

Abstract

Object-oriented programming (OOP) is a mainstream paradigm for engineering design software tool development. An emerging requirement is the introduction of semantics to achieve heterogeneous information sharing, but many challenges exist. Examples include using object methods to manipulate an RDF data, automatically converting data into RDF format, and supporting various programming languages. In addition, limitations to description capabilities for relationships among object-oriented classes exceed those of RDF, thus hindering direct mapping between object-oriented and Semantic Web classes. The proposed semantic object framework (SOF) combines object-oriented design and Semantic Web features. SOF utilizes embedded comments in source code to describe semantic relationships between classes and attributes. We use a mobile phone design case study to illustrate how the proposed system operates.

Keywords: Object-Oriented Programming, OOP, RDF, Semantic Web, MVC Design Pattern

致謝

一邊工作一邊唸書，對我而言是很寶貴的成長經驗，可以將社會大學體驗的東西分享到學校，又將學校習得的理論拿回工作上驗證，多年來的博士班生涯，我的指導教授羅濟群老師，就如同自己的父親一樣包容我，不但允許我有自由發揮的空間，又不時在旁給予各種人脈與資源的支持，對羅老師感恩的心，真是點滴在心頭。共同指導教授趙國銘老師是我的大貴人，讓一個原本完全不懂語意網的我，快速的引領我進入這個領域從事相關研究，很懷念趙老師總是在百忙之中抽空從英國Skype指正我的研究，雖然因為距離關係見面次數少，但每次碰頭，都像熟稔的大兄長一樣，提點著我前進的方向。感謝我的口試委員劉敦仁老師、林熙禎老師、黃興進老師、游張松老師，在口試過程中細心的給予我許多建議與指導。劉敦仁老師引領我進入推薦演算法的大門，並且對學生充滿耐心與謙和的風範，讓我十分欽佩。資管所游伯龍老師是我很感激的長輩，游老師的習慣領域學說讓我在最低潮的時候，重拾信心，更面對面的給我許多人生啟示，讓我銘感於心，非常溫暖。寫論文的日子當中，最支持我的莫過於我的太太宜敏，不只是給我精神上的打氣，宜敏運用自己的專業知識，給我許多過來人的建議，親自幫我校稿與訂正英文，並指導我跑統計的專業技術，在感情上與專業上都給了我最佳的支持。最近幾年唸書的日子當中，很幸運結識了鴻順（我的好友）、佩芸（我的伴娘）、其捷（我的伴郎），這幾位同學，用實際行動支援我，鼓勵我，也教導我，是我最棒的良師益友。資管所的淑惠，親切而溫暖的幫助我處理許多流程，很幸運有這麼棒的朋友。我在交大資管所從碩士班到現在，前後超過十年，感恩的話說不完，感恩的心停不了。

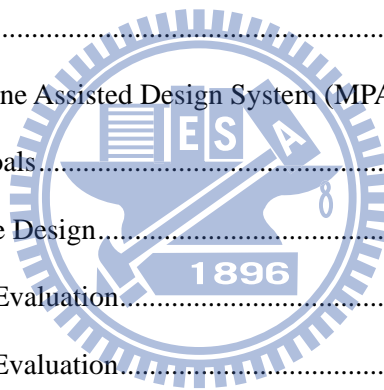
邱泊寰

中華民國九十八年十二月 謹於 交通大學

Index

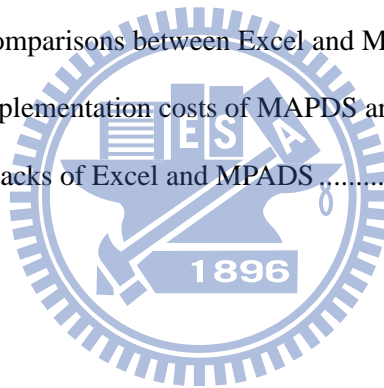
摘要.....	iii
Abstract.....	iv
致謝.....	v
List of Tables.....	viii
List of Figures.....	ix
Chapter 1. Introduction.....	1
1.1 Research Motivation.....	1
1.2 Research Goals.....	2
Chapter 2. Literature review.....	4
2.1 Semantic web.....	4
2.2 OWL language.....	11
2.3 Semantic web development tools.....	21
2.3.1 Jena.....	21
2.3.2 ActiveRDF.....	21
2.3.3 D2R.....	22
2.3.4 EClass.....	22
2.4 Model-View-Controller design pattern in OOP.....	22
2.4.1 Model.....	23
2.4.2 View.....	23
2.4.3 Controller.....	23
2.4.4 Object-Relational Mapping.....	24
2.4.5 Object-Semantic Mapping.....	24
2.5 Semantic Web Development Problems.....	25
Chapter 3. Semantic Object Framework (SOF) Architecture.....	28
3.1 Module Design.....	29
3.1.1 Data adapter.....	29

3.1.2	Parser	30
3.1.3	Query engine	31
3.1.4	RDF generator	32
3.2	Synchronization between class definitions and semantic descriptions	33
3.3	Implementation Details	35
3.4	Illustration using examples	40
3.4.1	Defining Address Book Classes Using OWL Syntax	40
3.4.2	Automatically Publishing Address Books in RDF Format	45
3.4.3	Making Queries Across Heterogeneous Address Books	46
3.4.4	Querying Data Sets with legal or illegal semantics	48
3.5	Discussion	50
Chapter 4.	Case Study	51
4.1	Mobile Phone Assisted Design System (MPADS)	51
4.2	MPADS Goals	52
4.3	Cooperative Design	53
4.4	Flexibility Evaluation	57
4.5	Efficiency Evaluation	59
4.6	Costs and Benefits Evaluation	62
Chapter 5.	Conclusions and Future Works	65
5.1	Summary	65
5.2	Future Works	65
References.	67



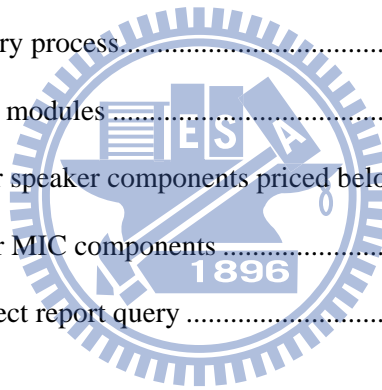
List of Tables

Table 1: The benefits of integrating O-O technology and the Semantic Web	3
Table 2: A comparison of functions for five Semantic Web development schemes. X denotes “unsolvable” and O “solvable”.	25
Table 3: The main implementation tools used by SOF	35
Table 4: Cooperative design features of MPADS.....	55
Table 5: Multiple roles in mobile phone design process	56
Table 6: A comparison of Excel, RDBM, and MPADS in terms of conditional query flexibility. X, unsolvable; O, solvable	57
Table 7: Query efficiency comparisons between Excel and MPADS	60
Table 8: Tasks for which Implementation costs of MAPDS are larger than a simple Excel file	63
Table 9: Benefits and drawbacks of Excel and MPADS	64



List of Figures

Figure 1: Five primary SOF modules.....	28
Figure 2: SOF data adapter class diagram.....	30
Figure 3: SOF parser class diagram	31
Figure 4: SOF query engine class diagram	32
Figure 5: SOF RDF generator class diagram	33
Figure 6: Jena separates semantic definitions from class definitions.....	33
Figure 7: SOF one-way synchronization process.....	34
Figure 8: How SOF supports the automatic conversion of data into RDF format.....	36
Figure 9: SOF semantic query process.....	38
Figure 10: Primary MPADS modules.....	53
Figure 11: Search results for speaker components priced below \$0.22 USD	59
Figure 12: Search results for MIC components	61
Figure 13: Results from defect report query	62



Chapter 1. Introduction

The research scope of the thesis includes the Semantic Web and the object-oriented programming. In this chapter, we discuss the research motivation and the research goals.

1.1 Research Motivation

As an evolving extension of the World Wide Web, the Semantic Web [Bemers-Lee, 01] uses semantic relationships among data to perform automated sharing and processing functions. Applications focus on process automation, data searches, data integration, and data reuse. Resource description frameworks (RDFs) [Lassila, 99] are used to represent Semantic Web data models. A basic RDF document contains statements consisting of a subject, predicate, and object. Engineers use this powerful representation tool to design processes and products to maximize knowledge and information sharing. Most existing engineering design tools are based on an object-oriented (O-O) paradigm, but the mismatch between O-O and the Semantic Web hinders the seamless integration of current design tools into Semantic Web based data models. Most software developers utilize the object-oriented programming (OOP) software design paradigm, but OOP is clearly unsuitable for processing Semantic Web data [Koide, 05][Koide, 06].

The most widely used function-dividing architecture for designing OOP classes is the model-view-controller (MVC) [Krasner, 88]. There are several object-relational mapping tools that can convert model objects associated with model classes into record formats for relational databases. Since RDF utilizes triple-oriented statements for data formatting, it differs significantly from MVC model classes. Furthermore, object-oriented classes cannot be used to describe semantic relationships among class attributes, thus making the task of converting model objects into RDF format for semantic queries more complex. Engineers have learned that the greater the amount of existing data requiring conversion into triple-oriented format, the greater the challenges in terms of performance and costs.

1.2 Research Goals

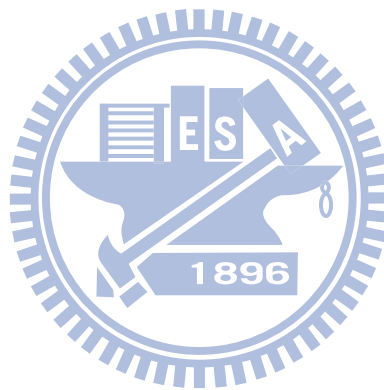
In this thesis we will describe a semantic object framework (SOF) for integrating O-O design with Semantic Web features. The benefits of integrating O-O technology and the Semantic Web are as follows (Table 1):

Benefits or Drawbacks	Description
Drawbacks of O-O without Semantic Web.	O-O technology hides semantic relationships in source code function data. Pure O-O technologies do not support data reasoning or inference as in Semantic Web technology. It is also hard for O-O to handle heterogenous data sources without Semantic Web technology.
Drawbacks of Semantic Web without O-O.	RDF data format is tedious with procedure programming.
Benefits of O-O + Semantic Web.	Object-oriented programming is mature, and many design patterns exist that can help programmers write reusable source code. The Semantic Web can publish information to the Internet as reusable data sources. It is a powerful means for integrating benefits from O-O (programmer-friendly coding style) and Semantic Web technology (machine readable

	web pages).
--	-------------

Table 1: The benefits of integrating O-O technology and the Semantic Web

The main goals are simplifying the tasks of (a) publishing model objects in RDF format via object-oriented design methods, and (b) making heterogeneous data queries in accordance with semantic relationships between classes and attributes. We use a mobile phone design case study to illustrate how the proposed system operates.



Chapter 2. Literature review

In this chapter, we discuss the background of our research, including concepts of Semantic Web, OWL language, semantic web development tools (Jena, ActiveRDF, D2R, and EClass), Model-View-Controller design pattern in OOP, and problems of semantic web development.

2.1 Semantic web

Semantic Web is invented to support a distributed Web at the level of the data rather than the presentation. Traditionally, one webpage could point to another page. Global references, also called Uniform Resource Identifiers (URIs), can be used to having one data item point to another. The Web infrastructure with semantic web technology can provide a data model to distribute information about a single entity. Meanwhile, it publishes a distributable, machine-readable description of the data, instead of only a human-readable presentation. The Semantic Web infrastructure uses a data model called the Resource Description Framework (RDF) to represent its distributed web of data [McBride, 02][Carroll, 04].

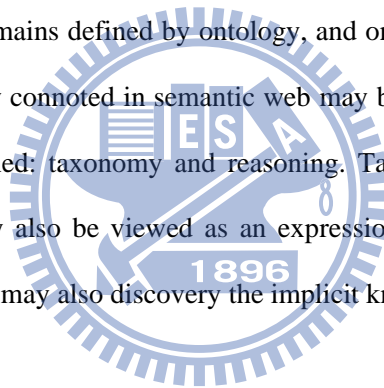
In the early 1990s, web resources are immediately and quickly constructed after Tim Berners-Lee developed World Wide Web (hereafter called WWW), and this is also known as a first-generation WWW. Some scholars proposed that we need a machine which can understand the resources in web at the time of resources getting bigger. Hence, Tim Berners-Lee also proposed another idea "Semantic Web" in recent years, and it is also called as second-generation WWW. Tim Berners-Lee defined this semantic web as "A web may be understood by machines", and it is also a collective of information. Since the goal of semantic web is to accomplish the targets of machine understanding, and understanding the meanings considerably close to reasoning to context.

In the architecture of semantic web, a layer of metadata is constructed on the WWW in order to describe the resources on WWW, such as HTML documents, image files, and others. Service functions provided to a user by

the portal of semantic web, such as inquiry, browsing, and service composition and among other things, are established on this metadata layer.

The resources of WWW are mainly used by human, and only human can understand the connotation of these resources, such as HTML documents, pictures or animations, and so on. These resources are not easily understood by computers according to present techniques. One task for computers can do is to visually present these specific format-based resource files to human for the purpose of interpreting the results, such as pictures and texts in HTML documents presented through browsers.

For accomplishing the purpose of semantic web, an adopted way utilizes the knowledge (including glossaries and relationships) used by different domains defined by ontology, and ontology is XML-based, and thus the web resources are easily accessed. Ontology connoted in semantic web may be applied to express web information so that two functions may be accomplished: taxonomy and reasoning. Taxonomy is a method for distinguishing different class information, and it may also be viewed as an expression of layer, while reasoning combines a relationship of both class and layer that may also discovery the implicit knowledge.



Machine readable

How does a computer read semantics? A computer should first utilize resource description framework (RDF) and Universal resource identifier (URI) linked to the related web page resources. The HTTP address used by everybody is an application of URI. Besides metadata, more and more people start using RDF to describe the knowledge contents connoted in web pages, and this is a big framework so that it is possible for one to search a specific resource over network. We dictate that everybody uses this method to describe your knowledge resource content, finds out your desired resources, utilizes ontology to define key terms through hypertext links, and makes logic reasoning.

A concept behind semantic network, widely speaking, is to use description language to describe any thing existed over network, and allows computer can "understand" what it is. For example, an object can be viewed as a part of car body or a person. If these objects can be identified, users may acquire enormous web data system links from computers. Owing to the high speed process abilities owned by computers, users may acquire enormous data, with the result of that the data obtained may be much richer than the ones derived from the results of human's unique brainstorming thinking. Therefore, scientists may apply this technique to develop new artificial intelligence (AI).

It is not just that a thinking machine should understand operations and logic rules, wherein much background knowledge should be involved. Before this, much knowledge should be entered into machines by human and specific formats and methods are needed for accessing data. But now, at the time of fully developed web, robots may also acquire information and apply them via web, and meanwhile, this has a closer relationship to the development of semantic web.

The present internet is still a human-based, and tens of thousands of web pages, texts, pictures, images, and others are presented and recorded using readable formats. But for a machine to interpret those things, it will not always be a piece of cake for an existing AI skill to do this. So, we can't directly ask questions to search engine; quite the contrary, we have to fuzzily search related information in accordance with search keywords. Because machines don't understand the contents of web pages, we thus have to compute these search keywords with statistics and scoring, and then rank these computed results.

In old web pages, miscellaneous tags are still needed to describe web documents, such as font, b, br, and others, for the purpose of beautification on browser; but in fact, they are nonsense to machines with the result of interpreting barriers. The idea of web page standard promoted by W3C intentionally separates the expressions from its contents, uses semantic markups to encapsulate these contents, and also applies CSS to control their appearance.

Furthermore, enough descriptions should be added into "human format" data, and then they can be read by machines so we can say that semantic idea exists. A layer of meaningful description added into semantic web in contents thus may allow machines to understand various data structures and relationships in contents in order that machines may process these data. Semantic web uses XML, RDF, OWL, or others as a structure, and thus they give assistance to data readable by machines. Moreover, these formats may not only restrict the application of web pages, but can also be used to exchange information between machines and understand it.

However, the development of semantic development is still at the very first stage. For web page support, it was a pretty hard thing to work on the conversion from old HTML to XHTML + CSS, not to mention both RDF and OWL (more support needed), so various changes will emerge during the evolution of semantics. In recent years, a new micro formats emerges, and it uses XHTML format and also can be embedded into existing web pages so as to carry out your site readable by human and machines. Machines may access data from micro formats in web pages, and know their meanings. Just because micro formats are small and exquisite, and integrated into web pages, it has had a high profile around the world, and is called as "Lowercase Semantic Web".

In any case, semantic web in AI is an important tool. Unlike a complex structure organized in human's brain, AI may easily be structured, and it may understand data structure, its meanings, and handle it. Using a common data format agreed among computers, and they then may know each other, and all systems may collaborate together for doing more things.

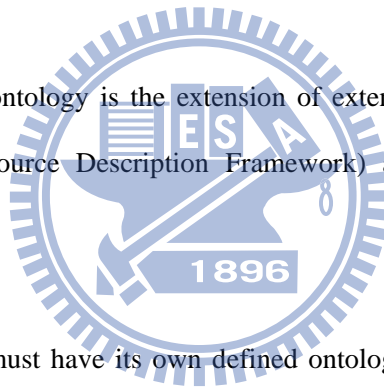
Ontology

A literal interpretation of ontology is knowledge of being. Ontology is knowledge to discourse things and investigate the essential of things. Ontology in computer science means a set of specific domain knowledge, these terminologies (glossaries) have distinct definition and description, and those may not only describe a certain idea in domain knowledge, but also elucidate the relationship between concepts.

In real world, each domain has a defined ontology, or ontology-based knowledge base. The same terminologies (glossaries) in different domains, times, and usages have different meanings. You may possibly acquire large amount of data in case of network search. Computer system doesn't know the domain a glossary belongs to so that searcher has to define the real meanings of this glossary, and its corresponding domain, and the relationship between glossaries.

Developing ontology should comprise four steps: define classes in ontology, define the layer-to-layer relationship between classes, define the attributes in classes, and describe the limitation to attribute values. After you follow above steps, the correspondingly specific entity for domain ontology can thus be established.

The architecture of currently used ontology is the extension of extended XML which adopts two ontology-based languages, such as RDF (Resource Description Framework) and (Resource Description Framework Schema), enacted by W3C.



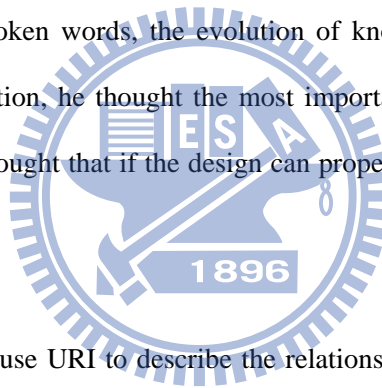
Each web page and each resource must have its own defined ontology, i.e. ontology-based knowledge base. The same glossaries used in different fields, times or usages may represent different meanings so that incorrect network search may usually occur in this case. Network doesn't know the domain for an glossary used in each web page, so searcher has to define the real meaning for an glossary, and the domain it belongs to. In any web page, ontology may tell you about the definition of each glossary, its corresponding knowledge scope, and architecture.

If any resource in web page has a declaration, and it tells the definition and architecture about the knowledge in web page to each visiting computer, then all visiting computers may read each web page.

We have mentioned a little about ontology which describes and defines resource knowledge content and information architecture of a web page. The idea of semantic web is that RDF may be applied to ontology or documents generated through similar programming language, and may clearly define conceptual relationship and reasoning logic rules. How to describe complete knowledge? We should tell computers about what we want to express essential data meaning, and this is for computers but not human so that you have to tell computers about part concepts and all concepts needed in this web page or this resource. Moreover, how to prosecute logic reasoning between concepts in computers? We first have to give computers an ontology definition, and then the logic reasoning could be prosecuted through this ontology.

Knowledge evolution

According to Tim Berners-Lee's spoken words, the evolution of knowledge is most important. Besides the ontology heavily used in web information, he thought the most important matter is the meaning existed in the evolution of knowledge, and he also thought that if the design can properly be taken, the semantic web is helpful in evolving human's knowledge.



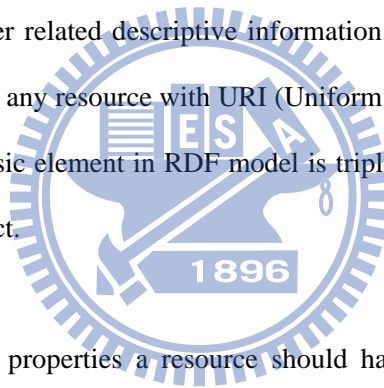
In each knowledge system, we may use URI to describe the relationship between concept and semantics, and then semantic web may help in doing the communication between concepts and the integration of knowledge systems. Since each knowledge system has its own architecture existed, the original conflict can thus be solved. If I tell you about my knowledge system and then you know my semantics and my reasoning obtained from this, the best communication can thus be well done after you first acquire my knowledge system.

Although the original design is to emphasize that the ontology is provided to computers, the bigger goal is that it is also to be recommended as a systematized reorganization of human's knowledge, and thus it makes the ontology readable by human, and also becomes a bridge for human's knowledge communication.

The most key issue the semantic web ideas should face to is: where can you acquire knowledge and its architecture, and how to construct it? Why each web resource (i.e., each web page) is stipulated in semantic web is to mark your own ontology in detail, and the starting point you will encounter is the variation and diversity between language glossaries and knowledge systems. The same things in different languages/dialects/domains have different names. The same nouns in different language contexts/usages/domains may have different meanings. An expression of concept can then be precisely interpreted until it knows the knowledge architecture behind the concept. This is the gap between information and knowledge that we need to stride across it.

RDF data format

RDF (Resource Description Framework) is a general-purpose description language used to describe the resource of World Wide Web and other related descriptive information. Applying simple and unified interface, you thus may use properties to describe any resource with URI (Uniform Resource Identifier) and the relationship between it and other resources. The basic element in RDF model is triple structure. Three major elements in this structure are Subject, Predicate, and Object.



RDF has no way to describe what properties a resource should have, and the relationship between these properties and other resources. RDFS (RDFS Schema) is a meta-data of RDFS, and its content defines basic glossaries used by RDF to describe resources.

Basic members of RDF architecture are resources and literals, and the relationship between members may be represented by additional tags with directional line. This looks like directed graph used in math. Resources in members may be used to represent a resource applied in WWW or an object which has no actual resource, and literals are used to provide factual data. A resource and another resource or literals and connection lines may be used to describe a fact, and it is equivalent to spoken sentence in our daily lives.

Information layer established based on RDF is a general relational data model that describes the relationship between resources or literals. These relationships are derived from the definitions of ontology-based knowledge base. An ontology-based knowledge base may collect entities and concepts in an application field, and classify them into different classification systems. Furthermore, characteristic for each class are also collected, and each of them describes types with respect to these characteristic and the relationship between them and other types, or the value corresponding to each character. In semantic web, XML syntax expression is used in ontology-based knowledge base, and its standard language used is Web Ontology Language (OWL , <http://www.w3.org/TR/owl-ref/>). When receiving a RDF document, the meaning of triple could be understood in accordance with the ontology-based knowledge base content referred by this document.

In contrast with relational database, ontology-based knowledge base is equivalent to the schema in relational database, and the RDF example generated based on relational database is equivalent to table data. The strong program service functions established over relational database should thanks to index of schema. Since reasoning abilities provided in ontology-based knowledge base are promoted to conceptual layer, the service of content-based retrieval is totally different from the one used in WWW.

Tim Berners-Lee has two ideal dreams about network. First one, he hopes every person may share knowledge through WWW, and the second one, he hopes computers may understand human languages, and the future network is a semantic web. The WWW established through URI (Universal Resource Identifier), HTTP (Hypertext Transform Protocol), and HTML (Hypertext Markup Language) proposed by Tim Berners-Lee has led to revolutionary change.

2.2 OWL language

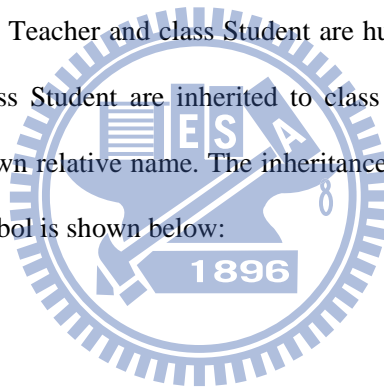
W3C enacted OWL language which is used to define the semantic relationship existed between semantic web data. However, owing to a large size problem existed in full version OWL Full that is impossible to figure out

meaningful values within a limited time, W3C has properly classified it into three versions: OWL Full (full version OWL), OWL DL (computer has ability to infer the computed semantic relationship in case of high speed computer operation), and OWL Lite (the simplest semantic relationship in case of low speed computer operation). Owing to the most important portion, OWL Lite, in OWL language, you may also understand which part in OWL you should handle first after understanding OWL Lite. Whereas, you should note that many limitations and simplifications still exist in semantic expressions of OWL Lite in contrast with OWL DL and OWL Full. Next, we will start by defining tags with respect to OWL Lite in RDF.

Class

A class equally defines a group of attributes and behaviors that should be included in a class, because of their common properties. For example, class Teacher and class Student are human class, and if we may define a class Human, because class Teacher and class Student are inherited to class Human, so that we may conclude that Teacher and Student must have their own relative name. The inheritance used here is represented by tag. C1 is a class declaration, and the available symbol is shown below:

```
class(c1)
```



rdfs:subClassOf

This tag represents the inheritance relation. For example, class "computerBook" is a subclass of class "book", and it is assumed that class Book has the property "bookName", and thus computer book and cook book have the same property "bookName" due to both of them inherited to class "book". This inheritance relation can be viewed as "a category of" relation, e.g., class "book" -> a category of "book". The c2 is inherited to the c1, and thus an example of symbol expression is shown below:

```
rdfs_subClassOf(c2,c1)
```

rdf:Property

This property tag is used to describe data values between individuals, e.g., hasChild, hasSibiling, hasAge. These properties represent "what children does it have" (a relation linked to the instance of class person), "what inherited relations does it have" (a relation linked to the instance of class Person) and "about their ages" (a relation linked to integer value type). p value of property x is y, and it's symbol expression is shown below:

$p(x,y)$

rdfs:subPropertyOf

This allows you to describe the inheritance between properties, e.g., if hasSibling may inherit to hasRelative, this means that if a relation hasSibling exists between two persons, they must have hasRelative relation. Its symbol expression is shown below:

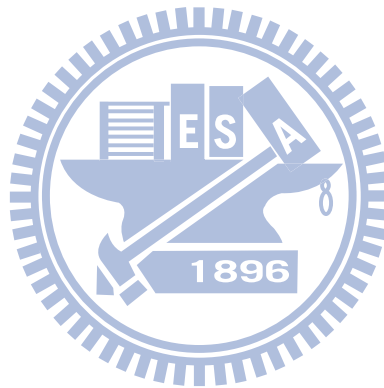
if

$\text{subPropertyOf}(p2,p1)$

$p2(x,y)$

then

$p1(x,y)$



rdfs:domain

This tag may restrict a object target of a certain property, and it is also subjected to a class. For example, dog an animal, so Dog rdfs:subClassOf Animal. If this dog belongs to a child, then this child must subject to the class Animal, and thus hasChild rdfs:domain Animal. This property value may be limited in a certain class.

rdfs:range

This denotes that if an object target of a certain property subjects to a class, then the usage of this tag is similar to that of rdfs:domain, but rdfs:range may also be additionally assigned. With respect to the range of it's property values, we, for example, may define the time to get doctor degree for some system, and thus we may assign this range within 1 ~ 8 years.

Individual

The individual is an instance of a class. For example, an individual called John effectively belongs to an instance in class Person. Next, we will individually introduce most important tags in OWL Lite.

owl:equivalentClass

This denotes that two classes are identical, e.g., class "student" and class "studentMentee" may be defined as the same semantics, and thus if you want to search related data about class "student" in case of deduction, data about class "studentMentee" may be also searched. For inference engine, this inference of equality relation may be applied to many identical things found from the source of heterogeneous data. An example of symbol expression is shown below:

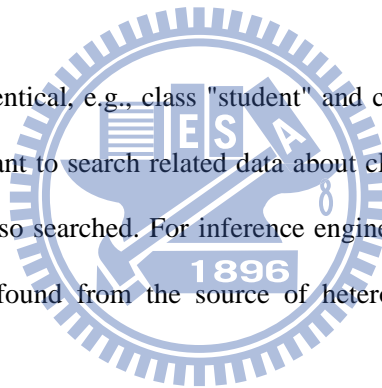
if

owl_equivalentClass(c1,c2)

is(x,c1)

then

is(x,c2)



owl:equivalentProperty

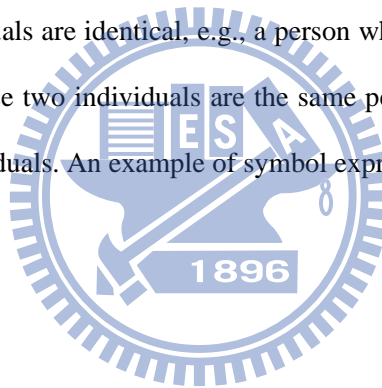
This represents whether two properties are identical semantics, e.g., property "hasGirlFriend" and property "belovedUnmarriedGirl" may have identical in semantics. An example of symbol expression is shown below:

```
if
  owl_equivalentProperty(p1,p2)
  p1(x,y)
then
  p2(x,y)
```

owl:sameAs

This represents whether two individuals are identical, e.g., a person whose English name is called John, and he has a nick name called Big J, then these two individuals are the same person, so owl:sameAs can be applied for the purpose of binding these two individuals. An example of symbol expression is shown below:

```
if
  owl_sameAs(x,y)
then
  x==y
```



owl:differentFrom

This may apply owl:differentFrom to identify whether these two different individuals are not identical in semantics. For example, if there two instances in class Person, one has property hasName assigned John, and the other one has property hasName assigned Johnny. Since these two classes Person looks alike, we here may conclude that they are different, and thus we may apply owl:differentFrom to identify that these two instances are not identical in semantics. An example of symbol expression is shown below:

```
if
  owl_differentFrom(x,y)
```


then

$x \neq y$

owl:AllDifferent

This tag is used for the purpose of specifying various different semantic instances. Except for verbosely using owl:differentFrom to specify every two different instances, you may have a simple expression, and thus you may utilize table to list all different instances, and apply owl:AllDifferent tag to specify a group of every two different instances. An example of symbol expression is shown below:

if

owl_AllDifferent(x,y,z)

then

$x \neq y$

$x \neq z$

$y \neq z$



owl:inverseOf

At property level which has various property characteristics may be used to describe the semantic relationship between properties. We'll introduce them as follows.

There is an existence of opposite relation between these two properties, e.g. let X hasChild Y is an existing fact, the system may automatically infer an opposite relationship of semantics if we have assigned a relationship owl:inverseOf to both hasChild and hasParent. For symbols used, if an opposite relation exists between p1 and p2, and p1(x,y) is true, then p2(y,x) may be inferred. An example of symbol expression is shown below:

if

owl_inverseOf(p1,p2)

p1(x,y)

then

$p_2(y,x)$

owl:TransitiveProperty

A property with TransitiveProperty characteristics may be assigned, e.g., a property blood relationship has such a feature, John has a blood relationship to Peter, and Peter has a blood relationship to Mary, then a blood relationship must exist between John and Mary so that such a relationship can be expressed by symbols, and if both $p(x,y)$ and $p(y,z)$ are true, then $p(x,z)$ may be inferred. An example of symbol expression is shown below:

if

$owl_TransitiveProperty(p)$

$p(x,y)$

$p(y,z)$

then

$p(x,z)$



owl:SymmetricProperty

This property has two-way relationship, e.g., if $p(x,y)$ expressed using symbols is true, then $p(y,x)$ may be inferred. For example, a friend relationship belongs to owl:SymmetricProperty, and if y is a friend of x, and thus we may also infer that x is a friend of y. An example of symbol expression is shown below:

if

$owl_SymmetricProperty(p)$

$p(x,y)$

then

$p(y,x)$

owl:FunctionalProperty

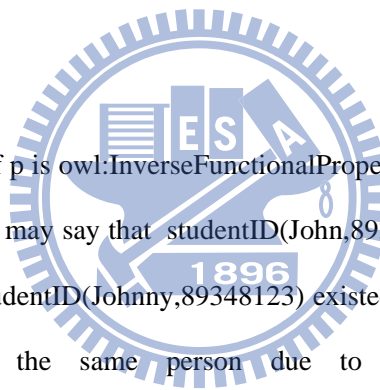
This property may only be assigned a single value, e.g., a property hasFather may have only one earthly father, and it is impossible for a person has two fathers in the real world, and thus we may say that hasFather must have only one value. By the way, the default value for owl:FunctionalProperty may be set to zero or one, i.e., NULL value is also permitted in this case. For symbol expression with $p(x,y)$, if x is fixed, then y is unique. An example of symbol expression is shown below:

if
 owl_FunctionalProperty(p)
 $p(x,y)$
then
 y is unique

owl:InverseFunctionalProperty

For symbol expression with $p(x,y)$, if p is owl:InverseFunctionalProperty, then x is unique while y is fixed. We here take student ID as an example, we may say that $\text{studentID}(\text{John}, 89348123)$ denotes that John's student ID is 89348123, and if we found a record $\text{studentID}(\text{Johnny}, 89348123)$ existed in our database, then we may conclude that John and Johnny must be the same person due to both student IDs has a property owl:InverseFunctionalProperty. An example of symbol expression is shown below:

if
 owl_InverseFunctionalProperty(p)
 $p(x,y)$
then
 x is unique



owl:allValuesFrom

This may invoke that any of property values is subjected to a certain class, or is limited within a range. For example, any value used in this property hasParent may be subjected to the one defined in class Human. An example of symbol expression is shown below:

```
if
  owl_allValuesFrom(p,c)
  p(x,y)
then
  y canOnlyBe c
```

owl:someValuesFrom

This may invoke that at least one of property values is subjected to a certain class, or is limited within a range. For example, at least one or more values used in this property hasParent may be subjected to class Teacher. On the other hand, this means that at least one or more parents subject to class Teacher. An example of symbol expression is shown below:

```
if
  owl_someValuesFrom(p,c)
  p(x,y)
then
  y canBe c
```

owl:minCardinality

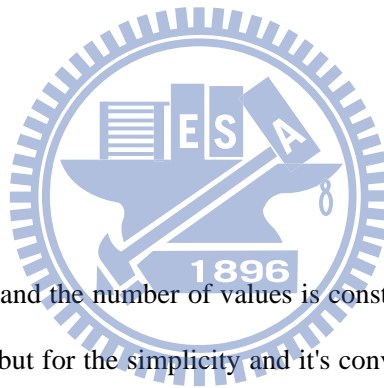
This allows you to restrict number of values you may use for this property. In OWL Lite, the value of minCardinality can only be set to 0 or 1, and if 0 is set, it means that this value is selectable, and the occurrence of it is not required while it is set to 1, the occurrence of it is required. An example of symbol expression is shown below:

```
if
    owl_minCardinality(p,1)
then
    number(p(x,?))>=1
```

owl:maxCardinality

This allows you to restrict number of values permitted for this property, e.g., each person may has only one nose, and thus this property may have at most one value. We take hasNose(John,aBigNose) as an example, owing to aBigNose assigned to John, he may not have another nose. An example of symbol expression is shown below:

```
if
    owl_maxCardinality(p,1)
then
    number(p(x,?))<=1
```



This may restrict it's property value, and the number of values is constant. This setting may set minCardinality and maxCardinality to the same value, but for the simplicity and it's convenience, owl:cardinality may be used to specify a constant number. In OWL Lite, it allows this number to be 0 or 1, and the example of symbol expression is shown below:

```
if
    owl_cardinality(p,1)
then
    number(p(x,?))=1
```

owl:intersectionOf

OWL Lite allows you to get intersection portion of two defined classes, and thus this portion can be given a new class, e.g., the Person is a class of a person, and EmployedThings is a class of employed things, and thus EmployedPerson has characteristics of these two classes. An example of symbol expression is shown below:

```
EmployedPerson==owl_intersectionOf(Person,EmployedThings)
```

or

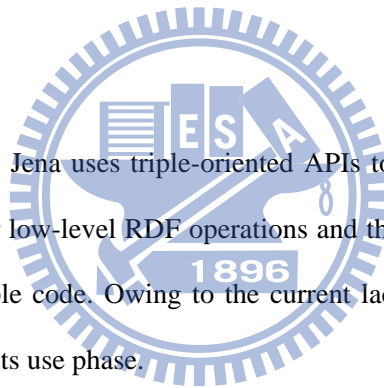
```
Woman==owl_intersectionOf(Human,Female)
```

2.3 Semantic web development tools

Before providing details of the SOF proposal, we will describe four Semantic Web solutions currently being used by developers and briefly review their positive and negative features.

2.3.1 Jena

Currently the most popular solution, Jena uses triple-oriented APIs to read/write and query RDF data. Jena's main advantages are its full support for low-level RDF operations and the fact that it is already in wide use, thus simplifying the task of obtaining sample code. Owing to the current lack of OOP integration, each operational step must be described in detail during its use phase.



2.3.2 ActiveRDF

This RDF object-oriented API is based on the Ruby language [Oren, 06][Oren, 07]. To perform the task of abstracting triple-oriented APIs, it uses O-O methods to manipulate RDF documents so as to simplify low-level API calling. Due to implementation limitations, this solution does not support the use of more than one programming language.

2.3.3 D2R

D2R directly converts relational database records to RDF format in order to facilitate RDF read/write and query functions [Bizer, 03][Bizer, 04]. Since the manipulated target is a database, D2R can be applied to any programming language and automatically perform format conversion (relieving programmers of this task) as long as the mapping relationship between database tables and RDF is clearly specified. Having a database as a manipulated target means that D2R does not support object-oriented encapsulation, thereby eliminating any possibility of data manipulation using objects.

2.3.4 EClass

This solution changes Java syntax to embed semantic descriptions into source code. [Liu, 04][Liu, 07]. EClass allows developers to define semantic relationships between attributes. However, an obstacle occurs when changing a widely used programming syntax, since syntax definitions affect existing programming tools such as compilers and virtual machines. Current programming tools need to be rewritten to support new syntaxes. Furthermore, the EClass solution currently lacks a query function for heterogeneous model objects.

2.4 Model-View-Controller design pattern in OOP

In the late 1970s, Model/View/Controller (MVC) concept was developed by the Smalltalk team at Xerox PARC to separate an application's data from the presentation of the data. In other words, the code to display the data does not mix with the code to compute the data.

In MVC terms, "model" stands for an application's data while "view" represents its presentation. The model and view belong to different parts of the code. For a large scale project, programmers with their special expertise could form corresponding teams to develop the model and the view. One important benefit is that each team only has to worry about their own problems when handling with issues.

The MVC is used to divide program codes into three independent types for the purpose of independent development, independent test, and code reuse. Because the main purpose of MVC is to achieve the complete independence of Model, class Model could neither apply any View nor introduce any Controller.

2.4.1 Model

MVC Model denotes data type. Owing to most data stored in database in computer business application, a way to map Model class in OO design to relational database becomes a very important automatic skill. Such a skill mapping Model data class to database is called "Object-Relational Mapping".

2.4.2 View

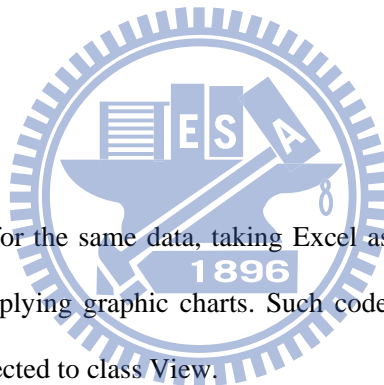
Owing to many presentation styles for the same data, taking Excel as an example, data may be presented by either using numeric number or by applying graphic charts. Such codes existed in a class that have ability to present original data onto GUI are subjected to class View.

2.4.3 Controller

Actual business operations logic should exists in Controller, such as payroll computation or report computation. A number of rule-based operations exist in such programs, and thus source code such as logic deductions or conditional operations and others should be ideally gathered in class Controller.

The dependency relationship of each other's MVC is shown below:

Controller->Model,View



View->Model

2.4.4 Object-Relational Mapping

Object-oriented design and relational database combining together has its advantage for development. Object-oriented programming languages have many reuse features, and they provide both inheritance and encapsulation. Owing to these, we may arbitrarily replace any low level data storage modules; and if we use object-based API to uniformly access data, we don't need to learn different SQL command operations due to different database supported by providers. For the purpose of thoroughly hiding low-level detailed operation of relational database, the best way is to directly map objects to data record of relational database in order to reduce the effort of transformation between them for program designers. We call this design of automatic transformation between objects and RDBM relational database as Object-Relational Mapping (abbreviated as OR Mapping).



2.4.5 Object-Semantic Mapping

Object-oriented design and semantic web combining together may speed up the development and also facilitate in the convenience of software development. Because a one-to-one mapping issue still exists between OO design and semantic web, many papers and discussions remain in this field, and all of these solutions aim at solving the automatic mapping from objects to semantic web. We finally hope that the appearance of any analogous OR Mapping tool kits may support this concern in order that program designers may select a suitable one solution from these, and also apply it to actual business applications or internet applications.

2.5 Semantic Web Development Problems

As shown in Table 2, there are at least seven problems associated with the integration of Semantic Web and O-O design:

Problem	Jena	Active RDF	D2R	EClass	SOF
Use object methods to manipulate RDFs.	X	O	X	O	O
Automatically convert data into RDF format.	X	X	O	O	O
Support various programming languages.	X	X	O	X	O
Use statements to describe class and attribute semantics.	X	X	X	O	O
Maintain semantic description files and class definition synchronization.	X	X	X	O	O
Support inheritance queries and heterogeneous data between classes and attributes.	X	X	X	X	O
Verify consistency in data and semantics.	X	X	X	X	O

Table 2: A comparison of functions for five Semantic Web development schemes. X denotes “unsolvable” and O “solvable”.

Using object methods to manipulate RDFs

Even though low-level RDF APIs provide complete RDF read/write and query functions, developers lack tools for utilizing objects to manipulate RDF data. As a result, development durations are longer, program codes relatively larger, and maintenance more difficult. The proposed SOF system uses O-O design to abstract RDF APIs to support the writing of program codes. Specifically, the system supports the use of O-O APIs for making queries, with corresponding query results returned in the form of model objects.

Automatically converting data into RDF format

Although some RDF APIs are capable of storing triple-oriented data for semantic query purposes, developers must convert model objects into triple-oriented format [Carroll, 03]—a detailed and time-consuming task. Thus, any development architecture capable of automatically converting model objects into RDF format will save developers significant amounts of time and effort. In addition, we have included an embedded web server that allows third-party software programs to use HTTP protocol to read RDF format data.

Supporting various programming languages

Instead of binding SOF syntax to a specific object-oriented programming language, we adopted a strategy of utilizing comments that describe class and attribute semantics to support the use of the SOF parser (with minimum modifications) with multiple programming languages [Kramer, 99][Leslie, 02]. Accordingly, programmers will only be required to learn SOF in order to develop applications.

Using statements to describe class and attribute semantics

The most straightforward way to combine Semantic Web and O-O design features is to describe class or attribute semantics, preferably at the same time that classes are defined. However, defining class and attribute semantics usually requires modifying programming language syntax. To address this modification issue without adversely affecting the original programming syntax, the proposed SOF system allows for embedded comments that support the limited use of RDF and OWL [McGuinness, 04] syntaxes.

Maintaining semantic description files and class definition synchronization

Some Semantic Web implementation solutions provide independent semantic description files that further modify relationships in existing data. This requires momentarily maintaining synchronous updates between files to prevent inconsistencies. Note that program API document and program code files are mutually independent and description document updates are frequently overlooked, resulting in obsolete and erroneous descriptions. JavaDoc uses embedded comments to prevent inconsistencies between API documents and program codes, which makes it easier for programmers to maintain consistency. The SOF solution is to apply similar principles to maintain program code and semantic description synchronization.

Supporting inheritance queries and heterogeneous data

Inconsistencies in column names across different databases are common (e.g., database A may use the term "Email" and database B "email"). To perform consistent queries involving all e-mails stored in two databases, the semantics of both terms must be clearly defined so that computers recognize them as equal. No architecture currently exists for defining semantic relationships between classes and attributes in OOP codes that allows a system to automatically acknowledge different attribute names with identical meanings. Problems also arise when performing unified queries of heterogeneous data sources. Model objects that result from queries may pertain to different classes, thus requiring a mechanism that allows OOP codes to distinguish among different classes of model objects and to manipulate attributes based on diverse classes. The proposed SOF system allows for the utilization of comments to maintain an inheritance relationship between attributes, and lets developers make unified queries of heterogeneous model objects.

Verifying consistency in data and semantics

Conflicts can occur between model objects and semantics. For instance, assigning an Email value to one unique account in an account management system can result in a later conflict when two accounts have the same Email value. Consistency in data and semantics requires a solution that can be easily applied. The proposed SOF system provides APIs for querying objects that developers can use to make semantic consistency checks.

Chapter 3. Semantic Object Framework (SOF) Architecture

In this chapter, we discuss the designs and components in the Semantic Object Framework (SOF), including SOF modules introduction, the design of modules, implementation details, and synchronization problems.

The five modules of the SOF architecture that address the above-listed problems are illustrated in Figure 1. The SOF data adapter reads data sources (i.e., CSV format files [Shafranovich, 05], database records, or proprietary data APIs) for conversion into model objects. Our system also allows programmers to write data adapters for other data sources. Model objects that represent SOF data adapter output include all data content (e.g., attribute values). Those objects later serve as input parameters for the SOF query engine and SOF RDF generator.

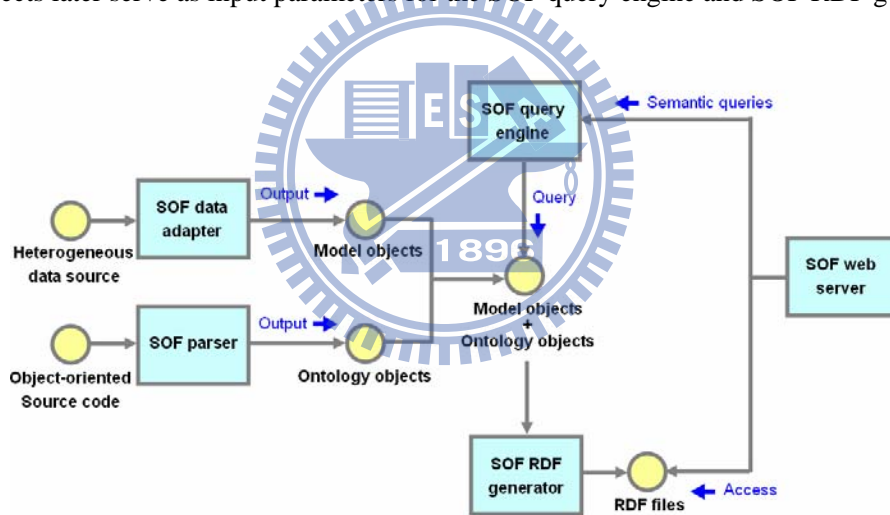


Figure 1: Five primary SOF modules

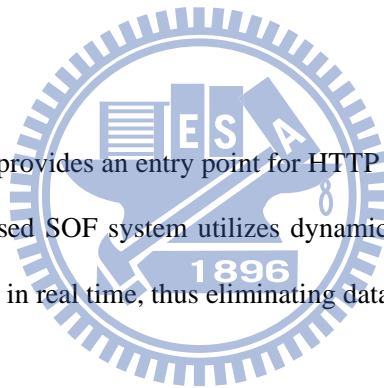
As its name implies, the function of the SOF parser is to parse SOF statements from comment lines in source code for the purpose of generating ontology objects, which include all information about semantic relationships between classes and attributes. The parser supports several of the most popular O-O languages, using a syntax that overcomes comment and descriptor variation problems. Module output consists of ontology objects in which

semantic class and attribute relationships are represented as objects. Ontology objects also serve as input parameters for the SOF RDF generator and SOF query engine.

The purpose of the SOF RDF generator module is to output model objects in RDF format so that third-party software programs can read RDF format data. Semantic relationships among model objects are recorded in the form of ontology objects that support RDF format file generation.

The SOF query engine module supports unified object-oriented API queries involving multiple heterogeneous data sources. Query results are presented as unified object arrays. Since returned model objects may be matched with different classes, APIs that are suitable for specific conversion types must be provided to address format conversion issues.

Finally, the SOF web server module provides an entry point for HTTP protocol so that third party programs can read RDF documents. Since the proposed SOF system utilizes dynamic conversion processes, all model object changes are updated to RDF documents in real time, thus eliminating data consistency concerns.



3.1 Module Design

3.1.1 Data adapter

The input terminal of this adapter is capable of handling several types of data sources. After performing model object format output conversions, object-oriented APIs are used to read and write model objects. The five SOF data adapters shown in Figure 2 are a DatabaseAdapter for reading records via database APIs, an RdfAdapter for reading data files in RDF format, a GmailContactAdapter for reading address book data via Gmail APIs, and a ThunderBirdContactAdapter for reading address books in ThunderBird data file format. Since these adapters are inherited from the SofDataAdapter class, they share common operation methods. Adapter output format is

presented as MVC model objects, which generally provide operation methods for reading and writing object attributes.

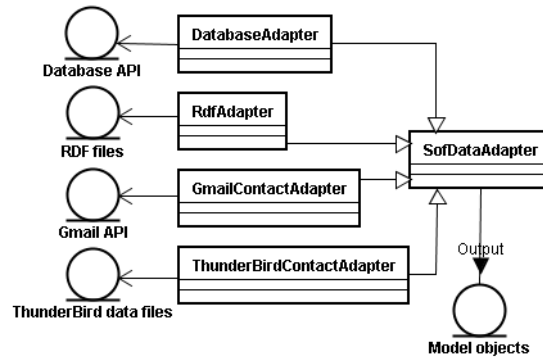


Figure 2: SOF data adapter class diagram

3.1.2 Parser

We have included three SOF parsers (Figure 3): PythonSofParser for reading Python code [Van Rossum, 03][Vrandecic, 05][Babik, 06], JavaSofParser for reading Java code, and RdfSofParser for reading class semantics in RDF file format. Since they are all inherited from the SofParser class, program code sharing is supported. Ontology objects generated by the SOF parser contain semantic relationships between classes and attributes. If ontology and model objects are used concurrently, heterogeneous data source semantic queries [Prud'Hommeaux, 06][Ying, 07] can be performed.

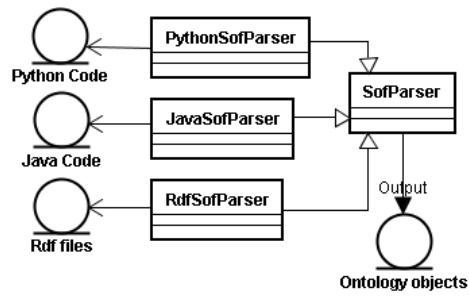


Figure 3: SOF parser class diagram

3.1.3 Query engine

Inputs consist of model and ontology objects. Our engine is capable of accepting query statements and outputting results in the form of model objects. The three SOF query engines shown in Figure 4 are a FilterSofQueryEngine for conditionally filtering semantic queries, a ValidSofQueryEngine for querying model objects that coincide with semantic rules, and an InvalidSofQueryEngine for querying model objects associated with illegal semantics. Since all are inherited from SofQueryEngine, all output results are presented as model objects.

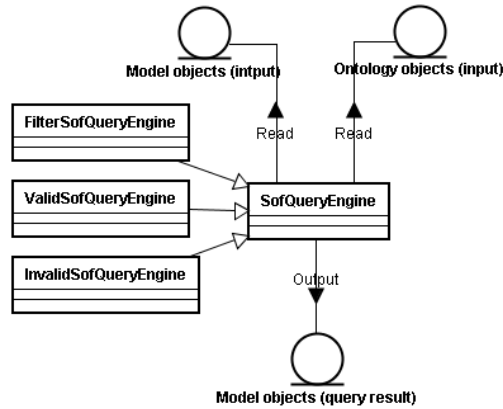


Figure 4: SOF query engine class diagram

For results generated as model objects by the FilterSofQueryEngine, only those that match query conditions are listed. During a query, developers can input object arrays for various classes, meaning that query results can also include different object classes. The proposed SOF system supports the use of APIs to obtain original model object class types; special processes can be used for different model object classes as necessary. For query results generated by the InvalidSofQueryEngine, model objects also include explanations for illegal objects—a useful tool for making corrections.

3.1.4 RDF generator

Generator inputs are model and ontology objects. The generator is capable of combining the two and outputting RDF strings, including semantic relationships between classes and attributes. As shown in Figure 5, final RDF string output can be stored in file format and accessed by other HTTP applications via the SOF web server. Since strings are expressed in standard W3C format and include model object data content as well as ontology object semantic relationships, any RDF format-capable application can be used to query and merge RDF strings.

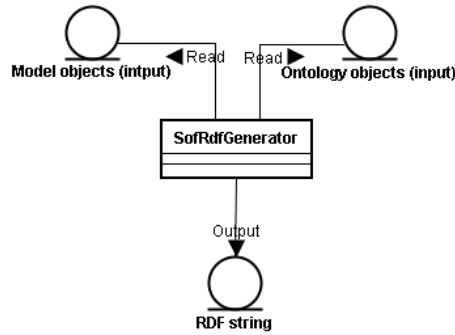


Figure 5: SOF RDF generator class diagram

3.2 Synchronization between class definitions and semantic descriptions

We compared Jena and SOF to illustrate why Jena is not a convenient means for maintaining synchronization between class definitions and semantic descriptions (Figure 6) as follows:

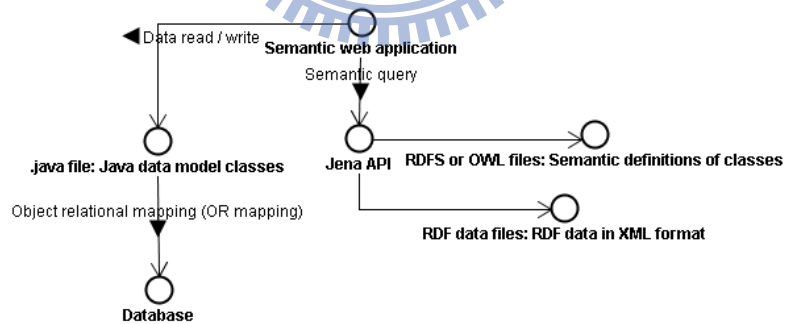


Figure 6: Jena separates semantic definitions from class definitions

A real-world Semantic Web application usually retrieves data from relational databases. The application accesses the database indirectly via object relation (OR) mapping and treats database records as Java model classes. The application can transform Java model objects into RDF format and perform semantic query operations. Before semantic queries can be processed, developers need to define semantic descriptions in RDFS/OWL format. Because .java files are separated from RDFS/OWL files, it is inconvenient to manually synchronize Java class definitions (in .java files) and semantic descriptions (in RDFS/OWL files).

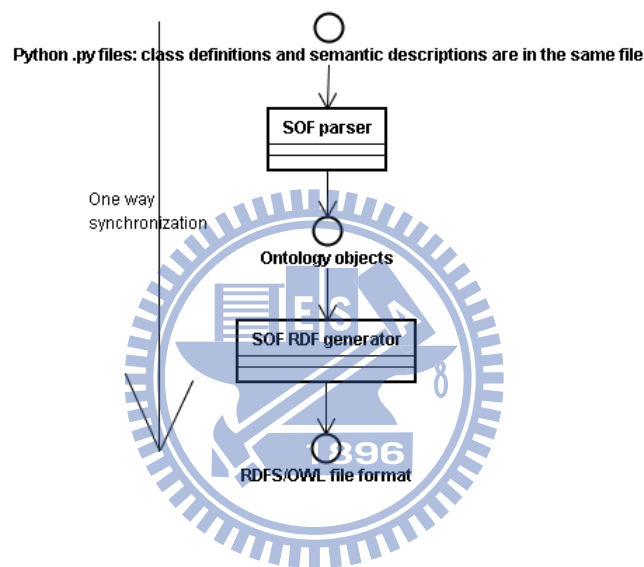


Figure 7: SOF one-way synchronization process

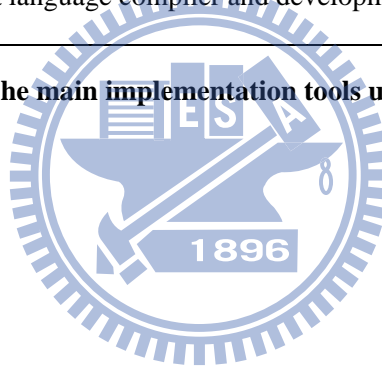
SOF applications allow class definitions and semantic descriptions to be held in the same file—a convenient feature for developers who want to keep them consistent in a text editor. For developers who want to produce RDFS/OWL files, the SOF parser and SOF RDF generator automatically read semantic descriptions from .py source code and output RDFS/OWL files. This one-way synchronization process (Figure 7) maintains consistency between class definitions and RDFS/OWL.

3.3 Implementation Details

The main implementation tools used by SOF are as follows (Table 3):

Tool	Version	Description
Python	2.4.6	Python language interpreter and run-time environment.
Django	0.96	High-level Python web framework that encourages rapid development.
Lighttpd	1.4	Lightweight HTTP web server.
Java	JDK 6	Java language compiler and development tools.

Table 3: The main implementation tools used by SOF



Sequence diagram (Figure 8) showing how SOF supports the automatic conversion of data into RDF format.

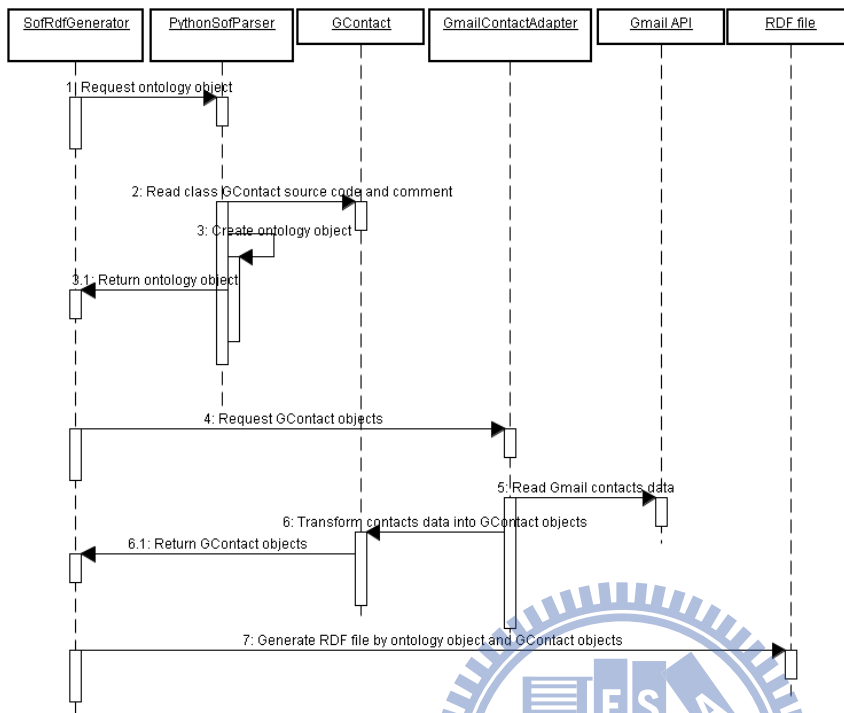


Figure 8: How SOF supports the automatic conversion of data into RDF format

We use a sequence diagram to help readers understand how SOF automatically generates RDF files from a data source. We use Gmail API as our data source for reading gmail contact information.

Request ontology object: SofRdfGenerator is responsible for initializing the RDF generation process. It sends initial requests to PythonSofParser and attempts to get ontology objects as return values.

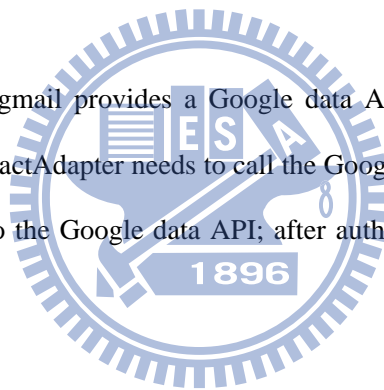
Read class GContact source code and comment: To produce ontology objects, PythonSofParser needs to parse python source code containing GContact class definitions and semantic descriptions.

Create ontology object: Ontology objects are dynamically created by PythonSofParser and preserved in Python run-time memory. Semantic relationships (represented by ontology objects) are like a directed graph data structure.

Return ontology object: After transforming embedded comments to ontology objects, PythonSofParser returns them to SofRdfGenerator.

Request GContact objects: SofRdfGenerator needs two input parameters to generate RDF files—ontology objects and model objects such as GContact. GmailContactAdapter receives requests from SofRdfGenerator and tries to return GContact model objects.

Read Gmail contacts data: Google gmail provides a Google data API to read contact information from its distributed network storage. GmailContactAdapter needs to call the Google data API. GmailContactAdapter sends a user's account name and password to the Google data API; after authentication, it can read the user's contact data.



Transform contact data into GContact objects: GmailContactAdapter transforms data from a Google data API to a GContact object. Data field names are mapped one-to-one. It is easy to transform data values as strings in GContact objects.

Return GContact objects: GContact objects are returned to SofRdfGenerator.

Generate RDF file by ontology object and GContact objects: After SofRdfGenerator receives both ontology and GContact objects, it gets all necessary information for generating RDF schema and RDF data formatting. The Django framework for our development tool provides a template architecture to dynamically generate files in any

format. SofRdfGenerator transforms ontology and GContact objects as string variables in a hashtable data structure, and then uses the Django template architecture to produce RDF files.

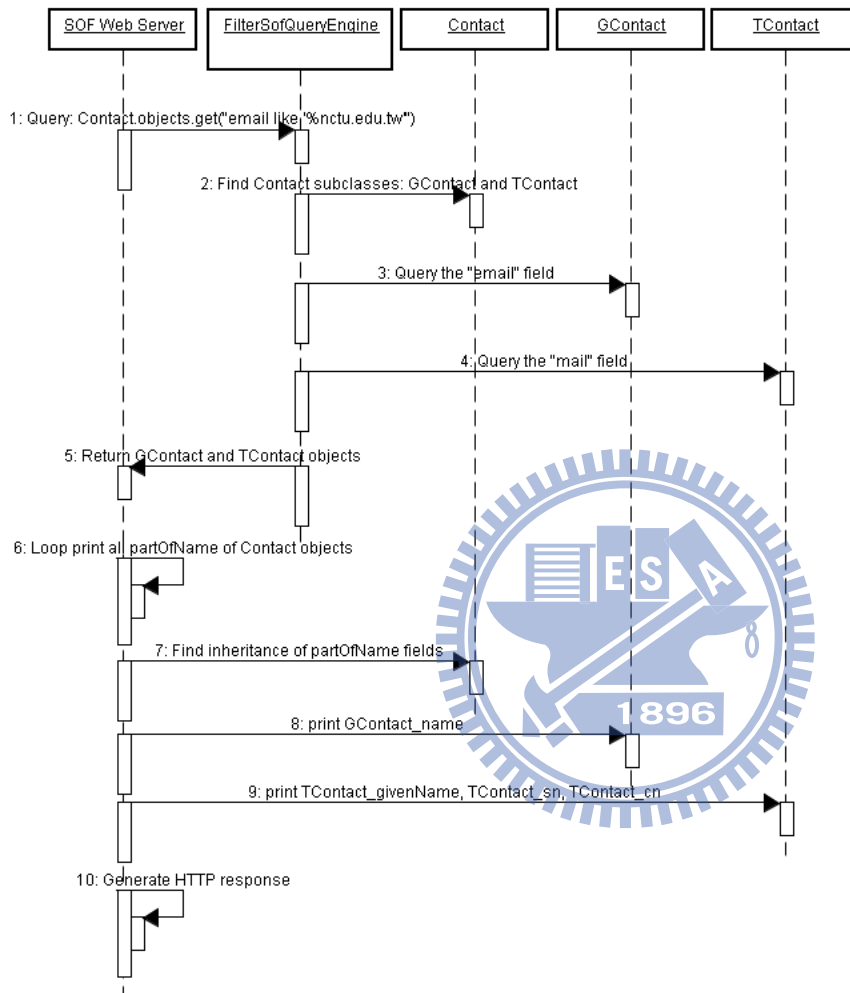


Figure 9: SOF semantic query process

We use a sequence diagram (Figure 9) to explain the SOF semantic query process and to show how the SOF query engine executes a semantic query and retrieves data from two separate sources such as Gmail and Thunderbird.

Query: `Contact.objects.get("email like '%nctu.edu.tw'")`: After semantic query strings are inputted in HTML format, the SOF web server receives a HTTP POST request from the web GUI and forwards the request strings to `FilterSofQueryEngine`.

Find Contact subclasses: `GContact` and `TContact`: At first, `FilterSofQueryEngine` only knows that the user wants to query all objects belonging to the `Contact` class and its subclasses; however, `FilterSofQueryEngine` does not know subclass names, all of which belong to the `Contact` class and are found in ontology objects. After searching `Contact` ontology objects and identifying the two subclass names `GContact` and `TContact`, `FilterSofQueryEngine` queries both subclasses and integrates results.

Query the `GContact` "email" field for matches to `"%nctu.edu.tw"`.

Query the `TContact` "mail" field for matches to `"%nctu.edu.tw"`.

Return `GContact` and `TContact` objects: All of these objects are added to a `Contact` data list and returned to the SOF web server to be presented on a screen.

Loop print all `partOfName` of `Contact` objects: SOF web server receives a `Contact` objects list and tries to print all contact names in a loop.

Find inheritance of `partOfName` fields: The `Contact` objects list has two subclasses. When the SOF web server tries to print `partOfName` fields for all `Contact` objects, those objects automatically locate all inheritance relationships by ontology objects.

print GContact_name: Contact objects determine that GContact_name matches partOfName fields and prints them out.

print TContact_givenName, TContact_sn, TContact_cn: Contact objects determine that TContact_givenName, TContact_sn, and TContact_cn all match partOfName fields and prints them out.

Generate HTTP response: The SOF web server integrates all returned values in HTTP response format and presents them via a web browser.

3.4 Illustration using examples

The purpose of this section is to illustrate the two primary functions of the proposed SOF system: (a) automatically converting model objects and publishing them in RDF format, and (b) performing semantic queries across heterogeneous data sources. The address book data used in the following examples are supported by Gmail and ThunderBird. Since they use different attribute names, under current conditions developers are forced to put a lot of time into format conversion to support queries involving both address books. Taking Python language as a specific example, the SOF approach is to add semantic relationships to classes and attributes when they are declared. After relationships are established, the two primary functions can take place.

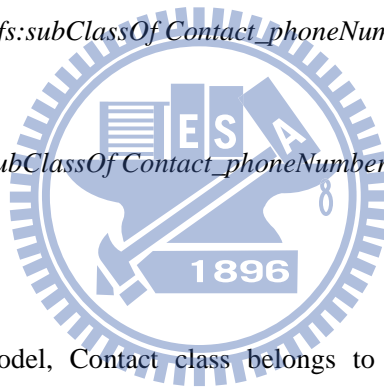
3.4.1 Defining Address Book Classes Using OWL Syntax

Before making a unified query across two address books, a user must first define a class named “Contact” for sharing common attributes. From a semantics perspective, this class is inherited to GContact (Gmail Contact) and TContact (ThunderBird Contact).

```

class Contact(Model):
    partOfName=""
    partOfAddress=""
    #owl:InverseFunctionalProperty Contact_email
    email=""
    phoneNumber=""
    #Contact_officePhoneNumber rdfs:subClassOf Contact_phoneNumber
    officePhoneNumber=""
    #Contact_homePhoneNumber rdfs:subClassOf Contact_phoneNumber
    homePhoneNumber=""
    #Contact_mobilePhoneNumber rdfs:subClassOf Contact_phoneNumber
    mobilePhoneNumber=""
    #Contact_faxPhoneNumber rdfs:subClassOf Contact_phoneNumber
    faxPhoneNumber=""

```



According to the MVC design model, Contact class belongs to the Model data class, therefore class Contact(Model) is declared as representing a Contact inherited to the Model class.

The presentation meaning of the “partOfName” attribute is a contact person's name, which contains a surname/middle name/full name/nickname, etc. Here we allow partOfName to represent a full name or any name segment. If the semantics of any other attribute are inherited to partOfName, the attribute is used to identify one contact person’s name string.

In Python, the pound sign (#) designates a comment. Since SOF syntax is embedded in comments, any instance of ‘owl:’ or ‘rdfs:’ included in a comment means the statement is SOF-specific. For example, ‘#owl:InverseFunctionalProperty Contact_email’ utilizes OWL syntax to modify its semantics, meaning that

Contact_email string values must be unique. This should not occur in cases where two different Contact objects have the same email attribute value. In situations where they have the same email string, the proposed SOF system identifies conflicting Contact objects and notifies programmers, who can apply various strategies to resolve the illegal semantics. OWL statements are helpful for programmers in terms of applying rich syntaxes to limit relationships between model objects.

E-mail attribute names differ across various applications. Examples in address book software programs include Email, email, mail, Mail, emailAddress, and EmailAddress—all with identical semantics. In order to display all attribute values for all emails across heterogeneous address books, all E-mail-related attributes must be inherited to Contact_email.

The next topic is the process through which GContact is inherited to well-defined Contact attributes.



```
#GContact rdfs:subClassOf Contact
```

```
class GContact(Model):
```

```
    #GContact_name rdfs:subClassOf Contact_partOfName
```

```
    name=""
```

```
    #GContact_email rdfs:subClassOf Contact_email
```

```
    email=""
```

```
    #GContact_phone rdfs:subClassOf Contact_officePhoneNumber
```

```
    #GContact_phone rdfs:subClassOf Contact_homePhoneNumber
```

```
    phone=""
```

```
    #GContact_mobile rdfs:subClassOf Contact_mobilePhoneNumber
```

```
    mobile=""
```

```
    #GContact_fax rdfs:subClassOf Contact_faxPhoneNumber
```

```
    fax=""
```

company=""

title=""

#GContact_address rdfs:subClassOf Contact_partOfAddress

address=""

The representative meaning of “#GContact rdfs:subClassOf Contact” is that the GContact class is semantically inherited to the Contact class, therefore if any object query commands are used to query all Contact model objects, the GContact object inherited to the Contact class will remain within the scope of the queried targets. In a later section we will show that TContact is also semantically inherited to Contact. Accordingly, when developers want to query model objects from two different address books (e.g., Gmail or ThunderBird), SOF automatically recognizes that both GContact and TContact objects must be involved within the query scope if Contact class is the target being queried. In this manner, the goal of querying heterogeneous address books can be easily accomplished.

According to the comment line “#GContact_name rdfs:subClassOf Contact_partOfName,” the name attribute in the GContact class is semantically inherited to the partOfName attribute of the Contact class. Thus, if developers specify the string value of the Contact_partOfName attribute that is being queried at a later time, the SOF system will also automatically query the string value of the GContact_name attribute.

GContact_phone refers to a multiple inheritance relationship. The attribute represented by GContact_phone can be a business or residence telephone. Since RDF syntax supports multiple inheritance relationships, SOF still allows for semantic multiple inheritance descriptions for classes or attributes. This is true even if the programming language (e.g., Java) does not support multiple inheritance relationships. Using GContact_phone as an example, regardless of whether a developer chooses Contact_officePhoneNumber or Contact_homePhoneNumber as a query target at a later time, SOF will always automatically query GContact_phone attributes.

The next example shows how TContact is semantically inherited to Contact:

```
#TContact rdfs:subClassOf Contact  
  
class TContact(Model):  
  
    #TContact_mail rdfs:subClassOf Contact_email  
  
    mail=""  
  
    #TContact_givenName rdfs:subClassOf Contact_partOfName  
  
    givenName=""  
  
    #TContact_sn rdfs:subClassOf Contact_partOfName  
  
    sn="" #first name  
  
    #TContact_cn rdfs:subClassOf Contact_partOfName  
  
    cn="" #full name  
  
    #TContact_telephone rdfs:subClassOf Contact_officePhoneNumber  
  
    telephone=""  
  
    #TContact_homePhone rdfs:subClassOf Contact_homePhoneNumber  
  
    homePhone=""  
  
    #TContact_fax rdfs:subClassOf Contact_faxPhoneNumber  
  
    fax=""  
  
    #TContact_mobile rdfs:subClassOf Contact_mobilePhoneNumber  
  
    mobile=""  
  
    #TContact_homeStreet rdfs:subClassOf Contact_partOfAddress  
  
    homeStreet=""  
  
    #TContact_mozillaHLocality rdfs:subClassOf Contact_partOfAddress  
  
    mozillaHLocality=""  
  
    #TContact_mozillaHState rdfs:subClassOf Contact_partOfAddress
```



```

mozillaHState=""

#TContact_mozillaHPostal rdfs:subClassOf Contact_partOfAddress

mozillaHPostal=""

#TContact_mozillaHCountry rdfs:subClassOf Contact_partOfAddress

mozillaHCountry=""

#TContact_street rdfs:subClassOf Contact_partOfAddress

street="" #street of company

#TContact_l rdfs:subClassOf Contact_partOfAddress

l="" #locality name of company

#TContact_postalCode rdfs:subClassOf Contact_partOfAddress

postalCode="" #postal code of company

#TContact_c rdfs:subClassOf Contact_partOfAddress

c="" #country name of company

```



The TContact and GContact classes are both semantically inherited to the Contact class. Many engineers know that this class is more complex than GContact from their experiences with various attributes pertaining to TContact—especially those referenced to addresses. No distinction exists between home and business addresses or among country, county, or street attributes. GContact only adopts an address attribute to represent all possible address strings. In TContact, nine attributes are referenced to address, all of them semantically inherited to Contact_partOfAddress.

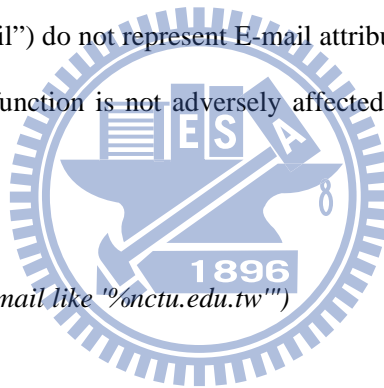
3.4.2 Automatically Publishing Address Books in RDF Format

Since HTTP access must be adopted using RDF format data, an SOF Web Server in the proposed system is responsible for providing a HTTP entry point; the corresponding model object RDF format can be accessed as long as its URL is appropriately entered (e.g., <http://localhost:8080/sof/Contact/>, which obtains RDF data

pertaining to Contact and its sub-classes). Developers wanting to access RDF for all model objects pertaining to GContact can visit <http://localhost:8080/sof/GContact/>; a similar URL can be accessed for TContact. Since the proposed SOF system uses an implementation technology to dynamically convert model objects to RDF format, developers have access to the latest data changes.

3.4.3 Making Queries Across Heterogeneous Address Books

The ability to make unified queries across heterogeneous database sources is an exceptionally useful Semantic Web function. Here we will give an example of code designed to find model objects pertaining to an email attribute ending with an “nctu.edu.tw” string from subclasses inherited to Contact. As noted in an earlier section, GContact (“email”) and TContact (“mail”) do not represent E-mail attribute names in the same manner. However, in the SOF system the unified query function is not adversely affected because they are inherited to attributes pertaining to Contact_email.



```
lstContact=Contact.objects.get("email like '%nctu.edu.tw'")
```

```
intCounter=0
```

```
for contact in lstContact:
```

```
    intCounter+=1
```

```
    print '=== Contact %s ==='%intCounter
```

```
    print 'partOfName:\n %s'%contact.partOfName
```

```
    print 'email:\n %s'%contact.email
```

These code segments will locate model objects with Email names ending with “nctu.edu.tw” from all classes inherited to Contact; the syntax for Contact.objects.get is similar to the SELECT command used in SQL—for example, “select * from Contact where email like '%nctu.edu.tw'.” Matching model objects may be in GContact

or TContact format. A “for loop” is followed, and partOfName and email attributes pertaining to the found model objects are displayed. The results are:

==== Contact 1 ====

partOfName:

"GContact_name": "Bowen Chiu",

email:

"GContact_email": "bowen@nctu.edu.tw",

==== Contact 2 ====

partOfName:

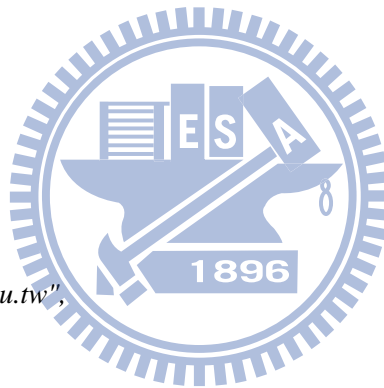
"TContact_givenName": "Kao",

"TContact_sn": "Gloria",

"TContact_cn": "Gloria Kao",

email:

"TContact_mail": "gloria@nctu.edu.tw",

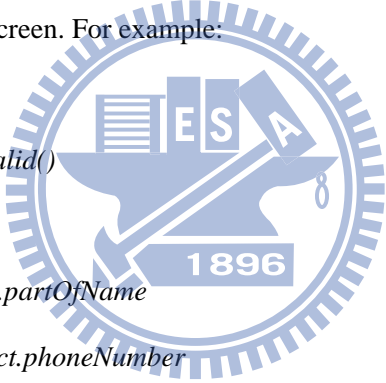


In this case, two model object records are displayed. Contact 1 pertains to GContact class model objects; the string value of contact.partOfName is "GContact_name": "Bowen Chiu". This leads to a key:value pair with ‘key’ as the GContact_name, which alerts developers that “Bowen Chiu” belongs to GContact_name. Contact 2 model objects belong to the TContact class, therefore the representative meaning of contact.partOfName is more complex. Here the value of contact.partOfName corresponds to an array delimited by a comma.

Developers who find it necessary to provide different data display methods for individual classes can use the SOF system to determine which class an object belongs to in accordance with a key:value pair associated with returned model objects. Accordingly, during a unified query, display formats for different classes can be adjusted if necessary.

3.4.4 Querying Data Sets with legal or illegal semantics

An RDF file may contain illegal data, thus requiring an effort to distinguish between legal and illegal data in certain situations. An example is finding non-duplicated mail name lists: since GContact and TContact probably contain duplicate person contact data, a previously defined SOF statement “#owl:InverseFunctionalProperty Contact_email” is required. In this statement, the limited Contact_email attribute value must be unique—that is, any two Contact model objects may not have the same Email value. If the same Email attribute value exists for more than two Contact model objects, from a semantics perspective they must be viewed as the same Contact object. Through this limitation, it is possible to utilize getInvalid() API to identify the model objects that violate this principle and to display them on a screen. For example:



```
lstContact=Contact.objects.getInvalid()
for contact in lstContact:
    print 'partOfName:%s'%contact.partOfName
    print 'phoneNumber:%s'%contact.phoneNumber
    print 'invalid reason:%s'%contact.getInvalidReason()
```

The first illegal data consists of

partOfName:

"GContact_name": "Bowen Chiu",

phoneNumber:

"GContact_phone": "+88635727001",

"GContact_mobile": "+886922387002",

invalid reason: validation fail->owl:InverseFunctionalProperty Contact_email

The second illegal data consists of

partOfName:

"TContact_givenName": "Chiu",

"TContact_sn": "Bowen",

"TContact_cn": "Bowen Chiu",

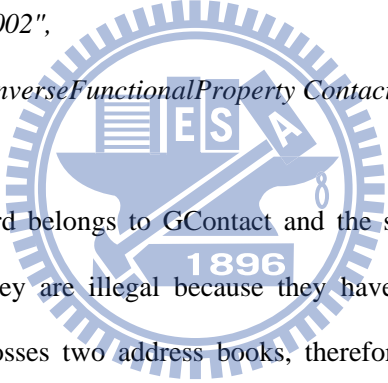
phoneNumber:

"TContact_telephoneNumber": "+88635727001",

"TContact_homePhone": "+88638885003",

"TContact_mobile": "+886993288002",

invalid reason: validation fail->owl:InverseFunctionalProperty Contact_email



The first illegal model objects record belongs to GContact and the second to TContact. Although they are considered different model objects, they are illegal because they have identical Email attribute values. The proposed SOF system successfully crosses two address books, therefore it is important that the function for finding semantically duplicated model objects is used to print a mailing list without duplications. The `contact.getInvalidReason()` command is capable of displaying the reason for a RDF semantic limitation violation; in response, developers can take such actions as deleting a redundant model object or merging two model objects into one. If a developer's intent is to use a command to read all legal model objects, "`lstContact = Contact.objects.getValid()`" can be used to add all legal model objects to the `lstContact` array—legal in the sense of Contact model objects with no duplicate Email attributes.

3.5 Discussion

The most significant benefits of embedded semantic comments into source code, is to expand semantic relationship for OOP classes and member variables without changing compiler or interpreter. Although comment style may cause typing errors during coding process, SOF parser can capture these errors and warning developer to correct them. The SOF provides an unified cross language architecture for coding semantic web in OOP environments.



Chapter 4. Case Study

In this chapter, we discuss the source code examples of SOF, and we use a mobile phone design problem to demonstrate how to implement a heterogeneous data query applications by SOF.

4.1 Mobile Phone Assisted Design System (MPADS)

Mobile phone design and manufacturing managers must work with component suppliers to create new products and systems. They must address such issues as component costs, compatibility, functionality, and capability. In this section we will discuss real and potential problems encountered in mobile phone design, show how the proposed SOF can be used to develop a mobile phone assisted design system (MPADS) to address them, and evaluate MPADS performance.

Mobile phone companies regularly manufacture and market multiple products concurrently. Product managers delay the need to design completely new mobile phones by referencing the component combinations of existing models—in other words, most successful designs can be reused and repackaged to create new phones with incremental specification changes. However, doing so raises challenges in terms of efficient information exchanges among independent design teams so as to achieve the greatest benefits from their different knowledge bases.

Here we will describe the case of a company using Excel files for purposes of documenting and sharing mobile phone specifications with design teams working in Taiwan, China, and Germany. According to current limitations, product managers wanting information for a specific component must manually open all Excel files and combine the required data into a new Excel spreadsheet. To support efficient knowledge sharing, we designed the proposed MPADS to produce efficient semantic queries without having to manually merge and edit files.

4.2 MPADS Goals

A mid-level mobile phone consists of between 50 and 60 components. During the design process, product managers must repeatedly perform design information queries based on previous experiences and product success. An efficient query system [Vega-Gorgojo, 08] allows product managers to make quick but informed decisions about new components and compositions. We therefore designed the proposed MPADS according to six goals: performing heterogeneous data queries; converting data to RDF format; converting component measurement units (e.g., speaker component dimensions) to fit query statements; analyzing mobile phone models and specifications based on required conditions; analyzing individual component specifications; and reviewing component defect reports.

During the mobile phone design process, a product manager will generally want to use the lowest price components that are sufficiently compatible. To accomplish this they must constantly perform data queries according to a complex mix of parameters. MPADS can help product managers perform such queries quickly and more efficiently than Excel files by using SOF development tools to define component classes and attributes in order to identify semantic relationships [Burger, 08] [Burkard, 08] [Valkeapaa, 08]. This process requires the conversion of Excel files (also referred to as comma separated values, or CSV) to model objects so that a programming language can directly read the information. Our SOF data adapter is capable of performing this reading/conversion task.

After creating classes and attributes from heterogeneous data sources, MPADS uses SOF syntax to define semantic relationships for further queries. Examples of component subclasses include Baseband, Display, Camera, MemoryNor, MemoryNand, MemorySram, MemoryCard, SawFilter, XCvr, Fem, Duplexer, Couplex, Pa, AnalogSwith, AudioAmp, ChargerIC, Comparators, EmiFilter, HallSensorIc, Ldo, LedDriverIc, LogicIc, XTal, PcbSingleLayer, PcbBuildUp, Fpc, Led, Mmp, Bluetooth, Gps, Wlan, FmRadio, IrDa, Led, Diode, Mosfet,

Mlcc, TanPolymer, Battery, Resistor, Inductor, Thermister, Varistor, EmiFilter, Fuse, Charger, Headset, Cable, Speaker, Vibrator, Receiver, and Microphone.

Data adapters use Excel CSV records as input and generate model objects as output for semantic queries. Readable outputs require a Customized Web GUI to convert text strings from SOF Web Server output format to HTML table format to help product managers compare component attributes. Figure 10 shows modules requiring developer implementation (grey background) and modules provided by SOF without additional programming requirements (white background).

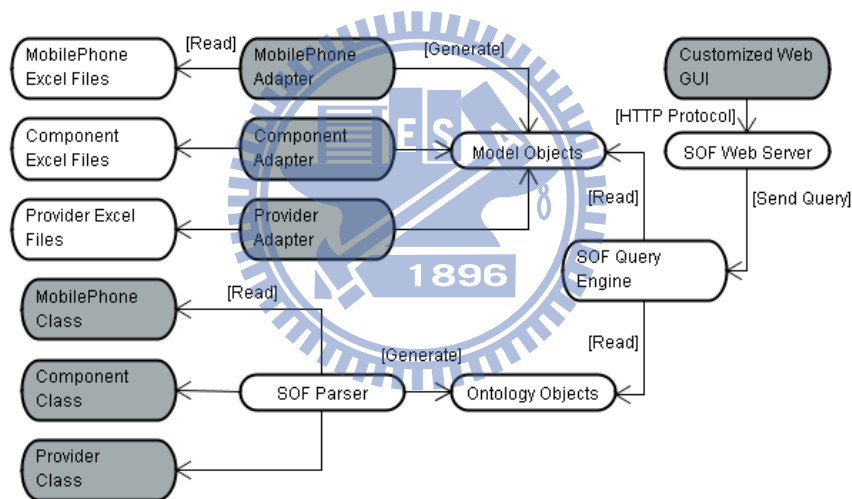
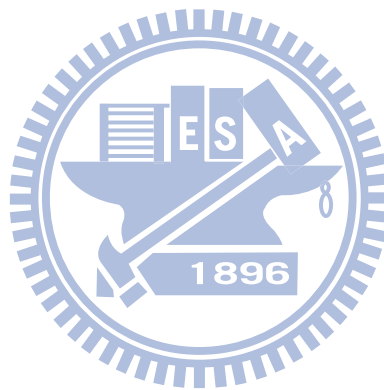


Figure 10: Primary MPADS modules

4.3 Cooperative Design

Our goal for MPADS is to help mobile phone designers working on a single sign on computer-supported cooperative system. We integrated MPADS with subversion (open source version control system) and mantis

(open source issue tracking system) projects. The following Table 4 presents the cooperative design features of MPADS:



Feature	Description
1. Knowledge sharing and semantic querying.	Product managers can perform conditional semantic queries to reference previous mobile phone design documents. Product managers can upload new designs to MPADS for sharing.
2. Structured component database sharing.	Mobile phone component specifications are originally stored in Excel or Word files without structure. MPADS allows designers to store structured information on component attributes in databases for spec. sharing purposes.
3. Design document co-editing.	Design documents can be uploaded, shared, opened, and edited by multiple users.
4. Online discussion.	MPADS users can post questions or share opinions online. Replies are collected in thread form and emailed to participating users.
5. Access control for user groups.	Users are divided into different groups. Each group has flexible access control as determined by an administrator. Design specifications are categorized to assist with controlling access.
6. Task assignments.	Managers can divide large design tasks into several subtasks and assign them to different developers. Priorities and task statuses can be monitored online by team members.
7. Merge modifications by version control.	If there are multiple users editing the same document, the version control feature can be used to solve collision problems via the automatic or manual merging of modification results.

Table 4: Cooperative design features of MPADS

For example, there are multiple roles [Aqqal, 08] in mobile phone design processes. MPADS allows for collaboration among various roles as shown in the following Table 5:

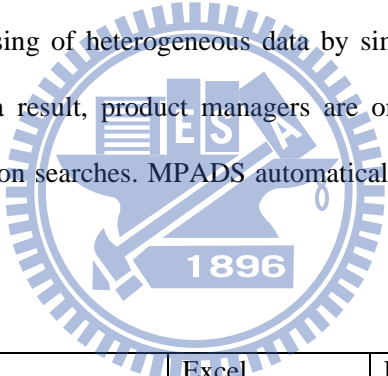
Role	Description	Cooperative Design Feature
Sales team	Collects customer feedback and new feature requirements from mobile phone markets. Posts market feedback on MAPDS and discusses feedback online with product manager.	4,5,7
Product manager (PM)	Coordinates business and technical teams with help from MAPDS. Responsible for tracking progress for new design and providing design specifications. Can use MPADS to perform semantic queries for hardware or software components.	1,2,3,4,5,6,7
Man-machine interface (MMI) team	Responsible for designing high-level software applications.	1,3,4,5,6,7
Layer 1 team	Provides application programming interface (API) for MMI team to control hardware functions.	1,2,3,4,5,6,7
Baseband team	In charge of mobile phone hardware layout and physical components. Can upload hardware component images and specifications to MPADS for users to perform queries.	1,2,3,4,5,6,7

Table 5: Multiple roles in mobile phone design process

4.4 Flexibility Evaluation

We evaluated differences among Excel files, relational database management system (RDBM), and MPADS in terms of query flexibility (Table 6) and efficiency. Regarding the first parameter, product managers generally specify component attribute values to perform conditional semantic queries. A drawback of Excel is the tendency for design teams to use different formats; this is especially true when those teams work in different countries, but it is not unusual among teams working for the same firm. As stated above, this requires the manual merging of query results into a new Excel datasheet, a time-consuming task. Developers who use RDBM cannot query heterogeneous data by simply applying SQL commands, since semantic relationships among database table fields require definitions.

MPADS allows for the easy processing of heterogeneous data by simplifying the task of defining semantic relationships for a body of data. As a result, product managers are only required to input single-line query commands to perform design information searches. MPADS automatically combines and presents search results in HTML.

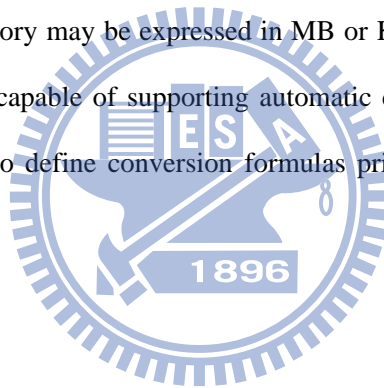


	Excel	RDBM	MPADS
Query heterogeneous data.	O (manually)	X	O
Convert data to RDF format.	X	O (D2R)	O
Convert component measurement units to fit query statement.	O (manually)	X	O

Table 6: A comparison of Excel, RDBM, and MPADS in terms of conditional query flexibility. X, unsolvable; O, solvable

Regarding RDF format conversion, a D2R system is available for automatically converting database records stored in RDBM format into RDF. While D2R is a convenient tool, it does not allow developers to manipulate data in an O-O fashion; lack of integration with OOP programming is its most significant drawback. The absence of OOP translates into more time required for product development tasks. MPADS lets developers define semantic relationships between classes and attributes, convert Excel files into model objects, and use an SOF Web Server to publish output in RDF format for reading by third party applications. It is equipped with OOP to reduce coding efforts, thereby releasing developers from having to write additional code for conversion tasks.

In the next area of comparison, mobile phone component attributes are frequently expressed in different measurement units—for example, costs may be expressed in US dollars or Euros, dimensions may be expressed in millimeters or inches, and chip memory may be expressed in MB or KB. Data stored in Excel format must be converted manually; RDBM is also incapable of supporting automatic conversions for measurement units. The proposed MPADS allows developers to define conversion formulas prior to performing queries. For example, Money class can be defined as



```
class Money:  
  
    intAmount  
  
    strMoneyType
```

Here *intAmount* represents quantity and *strMoneyType* a chosen currency. Using *Usd*, *Eur* or *Gbp* as *Money* subclasses, MPADS allows for value comparisons using a *MoneyConverter* class:

```
def getConverted(strSourceType,strTargetType,intAmount)
```

This method returns a converted currency quantity, *strSourceType* (representing the original currency type), *intAmount* (representing the original quantity), and *strTargetType* (representing the converted currency type).

Once the MoneyConverter class is implemented, MPADS uses a SOF query command to perform a search—for example:

```
Speaker.objects.get("price < Usd(0.22)")
```

This query finds all speaker components costing less than \$0.22 US, with prices for components manufactured in other countries automatically converted into a designated currency.

4.5 Efficiency Evaluation

Locating sources of less expensive components is a common product manager responsibility. An example of HTML query output is shown in Figure 11. Product managers can use this feature to compare component attributes from various suppliers by reading user-friendly HTML output. Efficiency comparisons for three related tasks are shown in Table 7. Note that RDBM was not considered, since SQL commands cannot be used to perform queries based on heterogeneous data.

Function	Speaker	Speaker
Dimension	16*3.3	16*4.1
Vender	PWS	CCA
Name	800-L7371	10-F-F8071SMD
P/N	23.4G652.002	23.4G940.002
Project	J9583	J2838
Picture		
Price	EUR 0.125	USD 0.21

Figure 11: Search results for speaker components priced below \$0.22 USD

	Excel	MPADS
Analyze mobile phone models and specifications based on required conditions.	1,219.46 secs	37.69 secs
Analyze single component specifications.	97.94 secs	13.88 secs
Review component defect reports.	46.28 secs	11.42 secs

Table 7: Query efficiency comparisons between Excel and MPADS

Using Excel files to perform manual queries requires locating strings in existing datasheets and cutting-and-pasting all matching data to a new datasheet. To determine the time required to complete this task, we performed each example query 3 times to obtain an average speed for finding information on 22 existing mobile phones and 140 components. For tests involving Excel, time was measured from the first opening of an Excel file to the completion of a datasheet. For MPADS, time was measured from the inputting of query strings in a customized Web GUI to the complete loading of a HTML result page into a browser.

Our tests were based on the knowledge that product managers are frequently required to perform conditional queries and to check component attributes. For example, in order to design a mobile phone that highlights multimedia functionalities, a product manager will likely perform at least three conditional queries regarding display size, camera resolution, and memory size. An example of a MPADS query command is

MobilePhone.objects.get('display.size > Pixels(120,160) and camera.megaPixels > MegaPixels(3) and internalMemory.size > MegaBytes(64)')

For designing and manufacturing a very slim mobile phone, an example of a MPADS query for MIC components is

Mic.objects.get('dimension < DimensionInMm(6.5,2.3)')

An example of results for such a query is shown in Figure 12.

Function	MIC
Dimension	4*1.0 with rubber boot
Vender	CIN
Name	BMG10BT
P/N	2C.42030.022
Project	J2738
Picture	
Note	rubber boot with coil spring

Figure 12: Search results for MIC components

Another common product manager function is checking defect reports as a means of avoiding unreliable components. In this case study, a defect was found in the handwriting display—it was incapable of capturing the correct coordination following a penDown event. To perform a MPADS query for defective component reports, a project manager would write

DefectReport.objects.get('component=Display')

An example of query results is shown in Figure 13.

Mobile Phone Model	J9882
Component Model	PSID283.4738.H
Defect Report Date	2008/3/1
Issues	Display component cannot capture the right coordinate when a penDown event is triggered.
Provider	TTC

Figure 13: Results from defect report query

4.6 Costs and Benefits Evaluation

Table 8 addresses tasks for which implementation costs of MPADS are larger than a simple Excel file. The two cost types are (a) static (one-time efforts during development cycle); and (b) dynamic (to integrate a new data source format into MPADS, developers must implement new classes).

Task	Description	Cost Type
SOF adapter	New data sources need new SOF adapters for reading and parsing into model objects.	Dynamic
Model class definition	New data formats need to define new model classes to represent them.	Dynamic
Semantic definition	New data sources need new semantic definitions in source code.	Dynamic
Customized web GUI	Web GUI differs according to the application	Dynamic

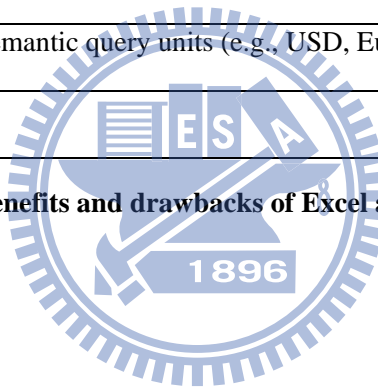
	being used.	
SOF parser	Individual programming languages need specific SOF parsers to process semantic definitions in source code. Once a SOF parser is implemented, it can be reused in all projects.	Static
SOF query engine	Can be reused in all projects.	Static
SOF web server	Provides HTTP access; can be reused in all projects.	Static
Training	Users need to be trained only one time to use SOF-based system.	Static
Server hardware	SOF system needs server hardware to provide web-based service to users.	Static
Server maintenance	SOF system needs an administrator to maintain proper function.	Static

Table 8: Tasks for which Implementation costs of MAPDS are larger than a simple Excel file

Excel files and MPADS have their individual benefits and drawbacks as follows (Table 9):

Benefit	Favored
Low static and dynamic costs for implementation.	Excel
Different departments can use unique data formats without extra communication, reducing overhead.	Excel
Low user-training costs.	Excel
No need for a hardware server to provide web-based service.	Excel
Ease and efficiency in querying heterogeneous data.	MPADS
Provides standard RDF formats for third-party data exchanges.	MPADS
Automatically transforms different semantic query units (e.g., USD, Euro).	MPADS
Various cooperative design features.	MPADS

Table 9: Benefits and drawbacks of Excel and MPADS



Chapter 5. Conclusions and Future Works

5.1 Summary

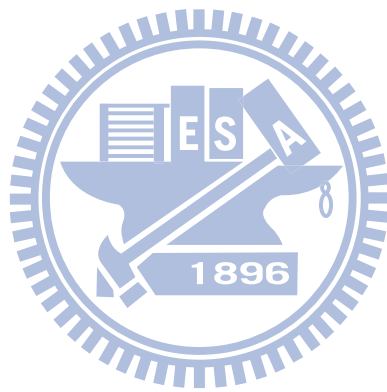
The focus of this thesis was on the publication of model objects to RDF documents that provide SOF solutions for automatic conversion tasks, as opposed to existing methods that require the manual conversion of model objects to a triple-oriented format. The proposed SOF system can be used to modify class and attribute semantics embedded in program code as well as to enhance descriptions of relationships between classes and attributes in object-oriented languages. In addition to preserving the synchronization of relationship descriptions between classes and program codes, the proposed system may support multiple programming languages. SOF provides a direct publication flow for the Semantic Web, allowing users to conduct queries across heterogeneous data sources and to incorporate positive features from both O-O programming and the Semantic Web.

Our main contributions are embedding semantic descriptions in source code without changing programming language syntax. Although EClass can also put semantic descriptions in class definitions, it changes the Java syntax and requires the rewriting of compilers. In a computer-supported cooperative work environment, it is very important to use developing tools with interoperability. SOF provides a better solution for developers to extend semantic features for existing object-oriented compilers or interpreters without rewriting them.

5.2 Future Works

The development tools associated with the SOF are insufficient, especially in terms of automation support for integrated development environment (IDE). An IDE development environment for various languages is required to support auto complete, dynamic syntax checking, and mutual synchronization between semantic diagrams and program codes [Dave, 02]. In cases where illegal SOF statement syntax occurs or where a semantic conflict between SOF statements emerges, a more powerful tool is needed to automatically analyze the problem and to

report results in a form that developers can use. In future projects we will work on IDE development tools to support the SOF system.



References

- [Aqqal, 08] Aqqal, A., Rensing, C., Steinmetz, R., Elkamoun, N., Berraissoul, A.: Using taxonomies to support the macro design process for the production of Web Based Trainings, *Journal of Universal Computer Science*, (2008)
- [Astels, 02] Astels, D.: Refactoring with UML, In *Proceedings of 3rd International Conference on eXtreme Programming and Flexible Processes in Software Engineering (XP2002)*, (2002) 67-70
- [Babik, 06] Babik, M., Hluchy, L.: Deep Integration of Python with Web Ontology Language, 2nd Workshop on Scripting for the Semantic Web (ESWC 2006), (2006)
- [Bemers-Lee, 01] Bemers-Lee, T., Hendler, J., Lassila, O.: The Semantic Web, *Scientific American*, 284, 5, (2001) 34-43
- [Bizer, 03] Bizer, C.: D2R MAP-A Database to RDF Mapping Language, *Proceedings of the 12th International World Wide Web*, (2003)
- [Bizer, 04] Bizer, C., Seaborne, A.: D2RQ-Treating Non-RDF Databases as Virtual RDF Graphs, *Proceedings of the 3rd International Semantic Web Conference*, (2004)
- [Burger, 08] Burger, T.: The need for formalizing media semantics in the games and entertainment industry, *Journal of Universal Computer Science*, (2008)
- [Burkard, 08] Burkard, B., Vogeler, G., Gruner, S.: Informatics for historians: Tools for medieval document XML markup, and their impact on the history-sciences, *Journal of Universal Computer Science*, (2008)
- [Carroll, 03] Carroll, J.J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., Wilkinson, K.: Jena: Implementing the Semantic Web Recommendations, Technical Report HPL-2003-146, HP Laboratories, (2003)
- [Carroll, 04] Carroll, J.J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., Wilkinson, K.: Jena: implementing the semantic web recommendations, *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, (2004) 74-83
- [Koide, 05] Koide, S., Aasman, J., Haflich, S.: OWL vs. Object Oriented Programming, *International Workshop on SemanticWeb Enabled Software Engineering (SWESE)*, (2005)
- [Koide, 06] Koide, S., Takeda, H.: OWL-Full Reasoning from an Object Oriented Perspective, *Asian Semantic Web Conf., ASWC2006*, 4185, (2006) 263-277
- [Kramer, 99] Kramer, D.: API documentation from source code comments: a case study of Javadoc, *Proceedings of the 17th annual international conference on Computer documentation*, (1999) 147-153

- [Krasner, 88] Krasner, G.E., Pope, S.T.: A cookbook for using the model-view controller user interface paradigm in Smalltalk-80, *Journal of Object-Oriented Programming*, 1, 3, (1988) 26-49
- [Lassila, 99] Lassila, O., Swick, R.R.: *Resource Description Framework (RDF) Model and Syntax Specification*, W3C Recommendation, (1999)
- [Leslie, 02] Leslie, D.M.: Using Javadoc and XML to produce API reference documentation, *Proceedings of the 20th annual international conference on Computer documentation*, (2002) 104-109
- [Liu, 04] Liu, F.F., Wang, J., Dillon, T.S.: An Object-oriented Approach on Web Information Representation and Derivation, *Proceedings of the 2004 IEEE International Conference on e-Technology, e-Commerce and e-Service*, (2004) 309-314
- [Liu, 07] Liu, F.F., Wang, J., Dillon, T.S.: Web Information Representation, Extraction, and Reasoning based on Existing Programming Technology, *Web Information Representation, Extraction and Reasoning based on Existing Programming Technology, Studies in Computational Intelligence*, 37, (2007) 147-168
- [McBride, 02] McBride, B.: Jena: A Semantic Web Toolkit, *IEEE Internet Computing*, 6, (2002) 55-59
- [McGuinness, 04] McGuinness, D.L., Van Harmelen, F.: *OWL Web Ontology Language Overview*, W3C Recommendation, (2004)
- [Oren, 06] Oren, E., Delbru, R.: Object-oriented RDF in Ruby, *Scripting for Semantic Web (ESWC)*, (2006)
- [Oren, 07] Oren, E., Delbru, R., Gerke, S., Haller, A., Decker, S.: ActiveRDF: object-oriented semantic web programming, *Proceedings of the 16th international conference on World Wide Web*, (2007) 817-824
- [Prud'Hommeaux, 06] Prud'Hommeaux, E., Seaborne, A.: *SPARQL Query Language for RDF*, W3C Working Draft, (2006)
- [Shafranovich, 05] Shafranovich, Y.: *Common Format and MIME Type for Comma-Separated Values (CSV) File*, Internet Eng. Task Force draft, (2005)
- [Valkeapaa, 08] Valkeapaa, O., Alm, O., Hyvonen, E.: An adaptable framework for ontology-based content creation on the semantic web, *Journal of Universal Computer Science*, (2008)
- [Van Rossum, 03] Van Rossum, G., Drake Jr, F.L.: *Python Language Reference Manual*, Network Theory Ltd, (2003)
- [Vega-Gorgojo, 08] Vega-Gorgojo, G., Bote-Lorenzo, M.L., Gomez-Sanchez, E., Asensio-Perez, J.I., Dimitriadis, Y.A., Jorin-Abellan, I.M.: Ontoolcole: Supporting educators in the semantic search of CSCL tools, *Journal of Universal Computer Science*, 14, 1, (2008) 27-58
- [Vrandecic, 05] Vrandecic, D.: Deep integration of scripting language and semantic web technologies, *ESWC Workshop on Scripting for the Semantic Web*, (2005)
- [Ying, 07] Ying, P., Tianjiang, W., Xueling, J.: Building Intelligent Information Retrieval System Based on Ontology, *Electronic Measurement and Instruments, 2007. ICEMI '07. 8th International Conference*, 4, (2007) 612-615