Vrije Universiteit Brussel – Belgium

Faculty of Sciences

In Collaboration with Ecole des Mines de Nantes – France

2002

Semantic Services Discovery in .NET

A Thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
(Thesis research conducted in the EMOOSE exchange)

By: Keqiang An

Promoter: Prof. Theo D'Hondt (Vrije Universiteit Brussel)
Co-Promoters: Jean-Marc Menaud (Ecole des Mines de Nantes)

ABSTRACT

Recent trends on ubiquitous computing have created new requirements for discovering and using the available services in the network. This leads to require semantic interoperability between heterogeneous entities, which means to realize dynamic ontology to exchange semantic information and configure automatically.

This thesis mainly concentrates on semantic discovery of service components. A survey of the state of the art in services discovery is provided, and two key procedures, dynamic adaptation and effective searches are being focused on. Having given a comprehensive survey of all major research being conducted in the area of ontology construction, the thesis proposes a generic approach to the semantic service discovery that allows establishing dynamic adaptive ontology by using RDF/S to facilitate the description of service components. And the prototype also provides the mechanism to combine the ontology infrastructure with conceptual retrieval module which is a powerful reasoning engine to acquire high performance on searching process. Conjunctive with Microsoft web service strategy, the architecture can be reflected on .NET platform eventually. Based on several foundation techniques an implementation has been developed for the architecture. Together with a handheld device application, a walkthrough scenario is performed to estimate the work.

ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to Professor Jean-Marc Menaud for his guidance and support in the preparation of this thesis. I would also thank him for believing my ideas and devoting substantial time to thesis-related-discussion. I would like to owe my special thanks to Zhao Liangjing and Marc Segura who are warm-hearted to give me precious advice during the early phase of the undertaking. I would also thanks to Zhen Hong who grants me such a great opportunity to study in EMN. Pure-hearted appreciation to Zhang Dongmei who companies with me and always brings joy to me so as to make me not feel lonely. And best wishes to my friends Lu Zhiqiang, Wang Xiangke, Song Wen, Zhuangli who share a good time with me during my stay in France.

TABLE OF CONTENTS

Chapter 1 Introduction

1.1 Objective

The purpose of this thesis is to design an architecture that uses a dynamic ontology on semantic description of components and services features, and make contributions to the service discovery recurring to logical concept retrieval. The main requirements for the architecture lie in two aspects; service discovering in distributed ad hoc network and the definition of automatic dynamic ontology. To capture the requirements and challenges of the related technologies, a comprehensive survey of research is conducted. The requirements are used to refine and select appreciate components and distributed computing technologies in the final implementation. An application for handheld devices applying has been built to showcase the potential of architecture for semantic services, and the implementation of design provides the facilities for adaptation and discovery of components and services in ubiquitous computing.

1.2 Motivation

Software complexity is linearly growing in connection with the increasing computing capacity of computers. Moreover since the expansion of networks, a new class of applications has emerged; in distributed applications, the treatment realized by the application is executed on different machine connected through a network. In this context, the growing usage of Internet gives birth to the concept of web application. Still problems need to solve to build such web application.

For example the construction of applications by using on-the-shelf components is still a research topic. It requires taking into account different, but essential concerns like: the methods and tools to build applications from on on-the-shelf components, the dynamic discovery of components based on semantic description of their features, and finally the techniques should allow dynamic adaptation of the generated application.

1.2.1 Scenarios

Let us illustrate the usefulness of service discovery with the following two scenarios; a handheld device in the network configuration and service registry update. These two cases are

survey in different perspectives, one is from the common user's point of view, and the other is from the network administrator's point of view.

A journalist reports with a sport event on the scene. He takes his personal digital assistant (PDA) in order to write a short essay and print it out, send mails, and get the response from the editor in the headquarter. He attaches his PDA to the local area network in the press room so as to access the Internet and use local resources such as monitors and printers.

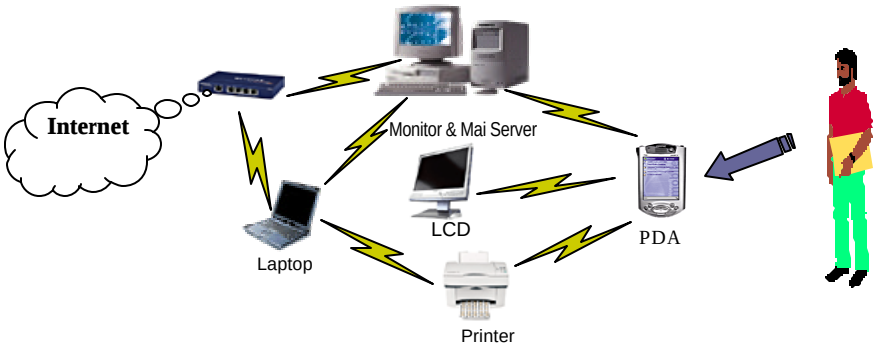


Fig. 1 A Ubiquitous Computing Scenario

We have a typical service discovery problem now: unless someone tells him the name and type of the printer and uploads the corresponding driver associating his handheld device, he will not be able to print out anything. He also does not know whether the printer supports color or not. Moreover, for Internet access he will have to re-configure his PDA with a valid IP address, DNS server (if he wants to access by TCP/IP). Furthermore, the email settings may be re-configured to the local mail server. The utilization of service discovery would enable him to automatically detect, select, and utilize these resources components and services. Service discovery would also inform him about the attributes of the monitors and printers, e.g. LCD pixels, paper format and color type.

A service repository executives update according to the new incoming components and changes on existing ones. The world is changing every time. The administrator needs to add the newly born service components into the registry and also mend the current schema, which used to descript the meta-data of the components and their relationships.

Obviously, the adaptive problem rises up. It will be common that the new component partly uses different ways to depict its features. And even more, the current met-data is unable to definite these feature structures, which means the schema is out of date. Further components ' characters may have conflicts to the current ones. The essential task for the administrator lies in how to make an adaptive approach to face the dynamic changing world. So the existing components can be mended and live safely with the subsequent ones.

In above two scenarios we come to think that from the user's point of view, service discovery greatly simplifies the task of finding and using services. While from the network administrators 'point, dynamically adaptability for components and services contributes to building and maintaining a service registry, especially to introduce new resources and services.

1.2.2 Requirements

In ad hoc environment, a prior information and description of the services components is, more often than not, unavailable. This is because the location of devices would be changing continuously. It is also impractical to assume that the mobile device would have an enumerated list of all possible services and their features. Therefore, a flexible service discovery infrastructure is an important base foundation for ad hoc environment.

While current existing service discovery technologies use simple interface-based and attribute-based matching. Service discovery is effectively done at a syntactic level. Actually, syntactic level matching and discovery is inefficient, which is due to the heterogeneity of service features in such a domain. For example, the same feature can be owned by different components but embodying distinct meaning in the context. This could result in the failure of syntactic match if the service query does not match with any component. Therefore, we need to discovery services in a semantic manner.

In the other hand, support in the exchange of data and information is a key issue in current computer technology. Ontology provides a shared and common understanding of a domain that can be communicated between user and application systems. Therefore, it may be play a major role in describing domain specific services. We can use the features of some ontology languages to reason about the capacities and functionalities of different service components.

So, the main requirements for the architecture lead to two aspects:

- ✧ The dynamic discovery of components based on semantic description of their features. This would involve with logical concept reasoning methods to enhance the efficiency of the discovery procedure.

- ✧ Definite a dynamic ontology that allows dynamic adaptation of the generated applications. This ontology should have the capacity of self-updating and self-mending to automatically fit for the dynamic changing repository of components.

1.3 Contribution

This thesis makes several contributions to the research area:

- ✧ It proposes a generic approach to the semantic service discovery that allows creating dynamic adaptive ontology to facilitate the description of service components.

- ✧ It combines ontology infrastructure with conceptual retrieval which is a powerful reasoning engine to acquire high performance on searching process.

- ✧ It provides a comprehensive survey of all major research being conducted in the area of ontology construction.

- ✧ It evaluates many potential technologies to realize service discovery upon different approach protocols.

- ✧ It proposes a feasible architecture model by making using of several existing technologies, and achieves module-independence in a certain extend.

- ✧ It provides a design and implementation for building .NET mobile device application to carry out a practical scenario to evaluate the performance on searching required service for handheld users.

- ✧ It creates a basic but refined service description schema for service providers to use to depict the features and characteristics for their own services components.

- ✧ It provides an implementation of the architecture and a walkthrough scenario described in the thesis.

1.4 Organization

This chapter introduces the scenario of the handheld application to definitude the field to dive in. Chapter 2 reviews the state of the art in service discovery for the thesis, including service discovery protocols and semantic service. Chapter 3 discusses the main focus on the related research. Chapter 4 illustrates the design for the overall architecture. The implementation for the design and a walkthrough scenario are showed in Chapter 5. Finally, conclusion and discussion are outlined in the last chapter.

Chapter 2 State of the Art in Service Discovery

This chapter will define the terminology and methodologies used in the future architecture. It will examine the different approaches to dynamic service discovery and semantic service description, and it will also explicit the future directions that exist within the domain. Specific research goals and contributions of this thesis will be selected in the next chapter based on the material presented in this chapter.

2.1 Terminology

In order to ensure the reader is familiar with the various terms used to describe dynamic service discovery, we will defined a few major compositional elements in this section.

2.1.1 Service Component

A service component is a self-contained body of code with a well-defined interface, attributes, and behavior. it is a specific kind of component which has been specifically designed to be reused or composed with other components. In other word, service components are the basic elements or building blocks that can be used to construct services. However, in order to simplify things, the terms component and service component will be used interchangeably throughout this thesis and they both refer to the definition provided below. A service component must have a name and related properties. The properties include a description of the component which may include operational constraints, its dependencies on other components or infrastructure, a list of operations that can be used or composed with other components, a description of the functionality of the component, a list of known relationships that it can be form with other components, and any other relevant information. The specification may also contain a description of the behavior of the service component using a formal language or structured syntax. The interface used to access the component may be described directly or indirectly in the specification. This definition is quite broad and thus allows a wide range of components to fall in its scope.

2.1.2 Service

A service, mush like a service component, is an entity that has a well-defined interface and behavior. The important characteristic that distinguishes a service from a component is its

visibility to the endpoint user. A Service can be referenced by a user and a service component cannot be directly referenced by a user. Individual components may also be classified as services if they meet the requirements of both definitions and thus may provide functionality directly to a user. In general, services are created by putting multiple components together using one or more mechanisms.

2.1.3 Service Discovery

Service discovery enable devices and services to exchange descriptions of components, i.e., advertisements, requests, notification evens, and responses to queries. From these messages, the participants will seek to select appropriate components, negotiate interfaces and parameters, and interact with the entities. The term "discovery" is used to refer to a spontaneous process, in which many entities discovery the other entities on the network, and present themselves to other entities. The services and devices must interoperate with other entities without pre-existing knowledge, and the configuration must automatically adjust to mobile and unreliable devices. So-called "discovery protocols" have the overall goal making digital network easier to create and use. The relevant protocols would be introduced in the following sections.

2.2 Techniques for Service Discovery

In this section a brief description of four existing industry supported protocols is presented. These protocols stand for the different views and approaches to make solutions for the practical problems [36]. Though, the core work for this thesis is not to find certain protocol to use but go beyond it to provide efficient query methodology, it would be helpful to know the background of the protocol infrastructure and look deep into the service discovery.

2.2.1 Protocols

2.2.1.1 Service Location Protocol

The Service Location Protocol (SLP) is a product of the SVRLOC Working Group of the internet Engineering Task Force [1]. It is a protocol for automatic resource discovery on IP networks. SLP is a language independent protocol. It bases its discovery mechanism on service attributes and can cater to any form of service, whether it is hardware or software.

The SLP infrastructure consists of three types of agents: *User Agent*, *Service Agent* and *Directory Agent*. The User Agents acquire service handles for end user applications that request for services. The Service Agents are responsible for advertising service handles to the Directory Agents. The Directory Agents collect together service handles and maintain the directory of advertised services. The core functionalities of the SLP are the following:

- ✧ Obtaining service handles for User Agents.
- ✧ Maintaining the directory of advertised services.
- ✧ Discovering available service attributes.
- ✧ Discovering available Directory Agents.
- ✧ Discovering the available types of Service agents.

A service is described by configuration values for the attributes which are possible for that service. For instance, a service that allows users to download audio or video content can be described as a service that is a pay-per-use real-time service or a free-of –charge service. The SLP also supports a simple service registration leasing mechanism that handles the cases where service hardware is broken but the services continue to be advertised.

2.2.1.2 Jini

Jini is a distributed service-oriented architecture developed by Sun Microsystems. Jini services can be realized to represent hardware devices, software programs or a combination of the two. A collection of Jini services forms a Jini federation. Jini services coordinate with each other within the federation. The overall goal of Jini is to turn the network into flexible, easily administrated tool on which human and computational clients can find services in a flexible and robust fashion. Jini is designed to make the network a more dynamic entity that better reflects the dynamic nature if the workgroup by enabling the ability to add and delete services flexibly.

One of the key components of Jini is the Jini Lookup service (JLS), which maintains the dynamic information about the available services in the Jini federation. Every service must

discover one or more JLS before it can enter a federation. The location of the JLS could be known beforehand, or they may be discovered using multicast. A JLS can be potentially made available to the local networks (i.e. the LAN) or other remote networks (i.e. the Internet). The JLS can also be assigned to have group names so that a service may discover a specific group in its vicinity.

When a Jini service wants to join a Jini federation, it first discovers one or many JLS from the local or remote networks. The service then uploads its service proxy (i.e. a set of Java classes) to the JLS. This proxy can be used by the service clients to contact the original service and invoke methods on the service. Service clients interact only with the Java-based services, both hardware and software services, to be accessed in uniform fashion. For instance, a service client can invoke print requests to a PostScript printing service even if it has no knowledge about the PostScript language.

2.2.1.3 Universal Plug and Play

Universal Plug and Play (UPnP), pushed primarily by Microsoft, is an evolving architecture that is designed to extend the original Microsoft Plug and Play peripheral model to a highly dynamic world of many network devices supplied by many vendors. UPnP works primarily at lower layer network protocol suites (i.e. TCP/IP), by implementing standards at this level. This primarily involves addition to the suite, certain optional protocols which can be implemented natively by devices. The keyword here is "natively". UPnP attempts to make sure that all device manufacturers can quickly adhere to the proposed standard without major hassles. By providing a set of defined network protocols, UPnP allows devices to build their own APIs that implement these protocols – in whatever language or platform they choose.

UPnP uses the Simple Service Discovery Protocol (SSDP) for discovery of services on IP networks. SSDP can be operated with or without a lookup or directory service in the network. SSDP operates on the top of the existing open standard protocols, using HTTP over both unicast UDP and mutlicast UDP. The registration/query process sends and receives data in HTTP format, but has special semantics.

When a service wants to join the network, it first sends out an advertise (or announcement) message, notifying the world about its presence. In the case of multicast advertising, the

service sends out the advertisement on a reserved multicast address. If a lookup (or directory) service is present, it can record such advertisements. Meanwhile, other services in the network may directly observe these advertisements. The advertise message contains an URL that identifies the advertising service and an URL to an XML file that provides a description of the advertising service.

When a service client want to discover a service, it can either contact the service directly through the URL that is provided in the service advertisement, or it can send out a multicast query request. While discovering a service through the multicast query request, the client request may be responded by the service directly or by a lookup (or directory) service. The XML service description does not play a role in the service discovery process [2].

2.2.1.4 Salutation

Salutation is a service discovery and session management protocol developed by leading information technology companies. Salutation is an open standard independent of operating systems, communication protocols and hardware platforms. The Salutation was created to solve the problems of service discovery and utilization among a broad set of appliances and equipment in an environment of widespread connectivity and mobility. The architecture provides applications, services and defines a standard method for advertising and describing their capabilities of others. The architecture also enables applications, services and devices to search for a particular capability, and to request and establish interoperable sessions among them.

The Salutation architecture defines an entity called the Salutation Manager (SLM) that functions as a service broker for services in the network. Services may be subscribed by meaningful functionality called Functional Unit, represent some essential feature (e.g., Print and Scan). Furthermore, the attributes of each Functional Unit are captured in the functional Unit Description Record. Salutation defines the syntax and semantics of Functional Unit Description Record (e.g., name, value).

SLM can be discovered by services in a number of ways such as the following:

- ✧ Using of a static table that stores the transport address of the remote SLM.

- ✧ Sending broadcast discovery query over the transport using the protocol defined by the Salutation architecture.

- ✧ Inquiring the transport address of SLM from a central discovery server. This protocol is undefined by the Salutation architecture. However, the current specification suggests the use of the Service Location Protocol (SLP) in conjunction with SLM.[3]

- ✧ The service specifies the transport address of a remote SLM directly.

The service process can be performed across the multiple Salutation Manager. One SLM can discover other remote SLM and determine the services registered there. Service discovery is performed by comparing a required service type, as specified by the local SLM, with the service type available on a remote SLM. Remote Procedure Calls are used to transmit the required service type from local SLM to the remote SLM, and then transmit the response from the remote SLM to the local SLM. Through manipulation of the specification of required Service type, the SLM can determine the characters of all the services that registered on a remote SLM, the characters of a specified service registered at a remote SLM, and the presence of a service on a remote SLM by matching a specified set of characters.

2.2.1.5 Deficiencies

Lots of papers have compared the above protocols, and take advantage of them and also inspect their born deficiencies in the field of service discovery. In this section, due to our main work is to present a dynamic ontology so we will illustrate the weakness but concentrate on the aspects which are related to our work, not for others [37].

Representation Deficiency

Components and services are heterogeneous in the nature world. These components and services are defined in terms of their own functionalities and capabilities. The functionality and capability description of these services are used by the clients to discover the desired services. The existing infrastructures of protocols lack expressive languages, representations and tools that good at representing a broad range of service descriptions and meantime offer a good reasoning mechanism to illation on the functionalities and capabilities of services [4].

For example, the service description protocols do not use any performance parameters for the existing services. They only find out a component or services and will not go beyond matching the query. Even more they do not consider whether the service would be able to server the requester properly.

Ontology Lack

Services in reality have no excuse to avoid interacting with the clients and other services. Service descriptions and information need to be understood and agreed by various parties. In other word, a well-defined common ontology must be present before any effective service discovery process can take place.

Actually, in most of the existing protocols the common ontology infrastructures are often missed or not well represented in the current service discovery architectures. Like Service Location protocol, Jini and Salutation do provide some mechanisms to capture ontology among services. However, these mechanisms like java interfaces and ad hoc data structures are difficult to be widely used by the industries to become standards. In UPnP, service descriptions are represented in XML which provide a good foundation for developing extensible and well-formed ontology infrastructure [5]. However, service description in UPnP does not play a major role in the service discovery process [2]. In the existing Jini architecture, ontology is captured in the level of Java interface types. While this unfeasible approach for developing common ontology, because it is difficult to be understood by the non-Java entities. Moreover, as a powerful programming language, Java is not suitable to describe the semantics inside the service and have limited abilities to capture the information of somehow complex services and components. Therefore, we imminently need of the expressive methods for ontology infrastructure. So RDF/S, a promising ontology description language, comes and turns out to be the core tools used in the implementation of this thesis.

2.2.2 The Semantics of Service

As it is argued in the previous section, the protocol models used by the industry standards do not meet the requirements that are stated. Discovery protocols enable devices and services to exchange descriptions of components, i.e., advertisements, requests, notification evens, and responses to queries [41]. From these messages, the participants will seek to select

appropriate components, negotiate interfaces and parameters, and interact with the entities. In each of these functions, it must be possible for the participants to adequately interpret the contents of the messages, even if some of the parties have never been encountered before. So it means the parties, or the descriptions of components must be semantically interoperable. Semantic interoperability requires that the parties that share a common model, which may be expressed in the one and more vocabularies. And it is possible they are using different mechanism to communicate with each other, and are mapped to the heterogeneous components of the system.

There are three main aspects should be focused on to comprehend semantically interoperability [38] [40]. The first is request filtering. One of the key goals of the discovery process is to locate instances of specifically desired of services. For instance, a personal digital assistant needs to find a display device nearby, with specified attributes, e.g. resolution 800 multiplies 600, to offer users a good view of some pictures rather than its small screen. To support this, a discovery service should provide "filtering" for service requests and notifications. So there must be a mechanism to constrain the results that match the needs of the entities. The greater the number and variety of services available, the more important of the filtering becomes.

Filtering amounts to the ability to process of querying to discovery the services, either from the user or from the applications or other program. The result of the query is some sort of list of services that match the query according to some rule. There are a great variety of possible approaches to filtering, with different kinds of queries, attributes, matching roles, and results.

Another aspect related to the issue is that when a service is identified to the client, what will the user receive? That is to say, what will the result of the query contain? For further spontaneous configuration, a very critical problem rises; after they discover each other, there must be enough common semantics to establish communication and negotiate interfaces and parameters. For example, considering a client requests a "printer" service, and receives a response containing an indicator to a kind of printer. How does the client know to use the printer? It is mot just a matter of locating the printer and its interface.

The third aspect is the service advertisement. When a device or service registers with the discovery service, what should it advertise? It ought to provide the information that will be needed to answer the queries, so the services need to know how to describe them that would be needed in terms that will allow the clients to discover it.

So in a word to summer up, the clients, services, and discovery service must have a common conceptual model of the devices, services, and their attributes. A specified protocol amounts to concrete realizations of a concept model of what kind of service and component may exist, and what client may ask for, what the query and response mean. The model must be shared by all the parities that need to communicate or participate in discovering. The model also must be extensible, to be able to incorporate new types of devices and clients. Additionally, the model must be fully automated, and kept up to update.

Though it seems difficult to conquer, an appropriate open standards expressed in XML formatted in the next chapter will begin to address the challenge.

2.3 Ontology and Data Schema

As it is hinted in the above, the terms that are expressed in the advertisements have to be defined in an otology. So before present the whole architecture of my contributions, it is necessary to outline the idea of ontology and state its relationship with XML-based schema, which is essential to my final implementation.

2.3.1 Concept of Ontology

Ontology defines the terms used to describe and represent an area of knowledge. Ontologies are used by people, database, and applications that need to share domain information (a domain is just a specific subject area or area of knowledge). Ontologies include computer-usable definitions of basic concepts in the domain and the relationships among them. They encode knowledge in a domain and also knowledge that span s domains. In this way, they make that knowledge reusable.

The word ontology has been used to describe artifacts with different degrees of structure. These range from simple taxonomies to meta-data schemes, or to logical theories. The

semantic discovery needs ontologies with significant degree of structure. These need to specify descriptions for the following kinds of concepts:

- ✧ Classes (general things) in the many domains of interest
- ✧ Relationships that can exist among things
- ✧ The properties (or attributes) those things may have

Ontologies are usually expressed in a logic-based language, so that detailed, accurate, consistent, sound, and meaningful distinctions can be made among the classed, properties and relations. Some of the ontology tools can perform automatic reasoning by means of ontologies, and thus provide advanced services to intelligent applications, such as conceptual semantic search and retrieval. Ontologies can prove very useful for a community as a way of structuring and defining the meaning of the meta-data terms that are currently being collected and standardized. Using ontologies, the applications can be intelligent than ever, in the sense that they can more accurately work at the human conceptual level.

2.3.2 Data Schema

Data schema has been developed in the computer science to describe the structure and semantics of data. A database schema defines a set of relations and certain integrity constraints. A central assumption is that the atomicity of the elements that are in certain relationships. In a word, an information source is viewed as a set of table, diagram, or any other kind of modality. However, many new information sources now exist that do not fit into the rigid schema. Therefore new schema languages have arisen to better fit for the need of richer data models.

Basically, they integrate schema for describing documents, such as XML, with meta-data designed for describing data. A prominent approach for a new standard for defining schema of rich and semi structured data sources is XML schema [6][7][8]. It is a means for defining constrains on valid XML documents, and provide basic vocabulary and predefined structuring mechanisms for providing information in XML.

2.3.3 Relationship

Ontologies applied to information source may be seen as explicit conceptualizations (i.e., meta information) that describe the semantics of the data. In [9] points out the following differences between ontologies and schema definitions:

- ✧ A language for defining ontologies is syntactically and semantically richer than common approaches to data sources.
- ✧ The information that is described by an ontology consists of semi structured natural language texts and hot tabular information.
- ✧ An ontology must be a shared and consensual terminology because it is used for information sharing and exchange.
- ✧ An ontology provides a domain theory and not the structure of a data container.

These statement need to formulated more precisely when comparing ontology language with XML schema language and the purpose of ontologies with the purpose of meta-data. More explication would be presented in the following chapter when talking about RDF Schema, the one used in the final implementation.

2.4 Research Ideas and Unsolved Problems

There are many research problems left to attack in the domain of dynamic service ontology and service discovery. Some of these problems will be addressed directly in this thesis. Others are topics for future work. Here is a list of some of the outstanding problems that need to be addressed in the problem domain that have been generated from the survey of related work.

- ✧ A design for a dynamic service ontology infrastructure that can adapt to a wide range of services components, and have the ability to distinguish the difference among them.
- ✧ The ontology allows re-mending and self-updating functionalities so as to realize the adaptation.

- ✧ A design for referencing and reasoning in the existing knowledge base, and offering a high performance for the querying.

- ✧ A design for linking the semantic discovery part with the concept retrieval part, and a mapping mechanism is also needed.

- ✧ A handheld application that justifies the use of service discovery must be devised. So that a practical scenario could be realized and help to estimate the architecture developed.

Now that we have a handle on the research being conducted in the dynamic service ontology and service discovery, we will attempt to define a focus for research for this thesis.

Chapter 3 Focus for Research

After present an overview of the semantic service discovery, in this chapter we would concentrate on the main problem domains and illustrate the approaches that meet the requirements mentioned before and find the efficient solutions to the problems.

3.1 Research Direction

As the requirements outlined in the first chapter, there are two main problem domains; dynamic adaptation and efficient discovery. Since they lie in different aspects to be surveyed on, normally, we divide them into two different procedures to make progress.

3.1.1 Adaptation Procedure

In the process of adaptation, we must find a mechanism that has the capacity of self-updating and self-mending to automatically fit for the dynamic changing repository of components. That is, we could describe the process as follows.

A repository should be established first to play the role as the knowledge base for the further use, such as query, discover and update. It can be either a text flat or a database supported. Then a basic data schema is stipulated for describe the meta- data, that is, the sharing conceptualization, which realizes interoperability among different components. The next step is that each of the services and components are described by some uniformed languages so convenient as to interact with each other, moreover to interact with repository. With the consistent information exchanged between the data schema, the meta-data can be intelligently updated and re-mended. In that case, the new coming up services and components would easily and safely involve into the system, and with no need to change itself descriptions when meet certain conflictions with others. And the last step is the repository arranges all the related information, that is, the descriptions of all the existing services, and ready for the clients or end point users to query and look up what they want.

Towards this procedure, the research direction is focus on finding good ontology descriptive methods in order to construct a dynamic and flexible ontology schema, which can deal with the changes even and again. For example, the facets lie in how to figure out a new

heterogeneous component, how to depict it, how to reflect these remodels to the end users and so on.

3.1.2 Discovery Procedure

The procedure for discovering services and components is the other research domain to be focus on. In this procedure, since the search action is inside the repository, the way how the look-up and discovery processes executive is much associated with the way how the repository is constructed. As we use data schema to constitute the concepts which actually we use RDF Schema in the final design, so the main problem is specified that is to find an efficient discovery mechanism in the RDF data infrastructure. Due to the workload, this mechanism could be an optimization on the query information or the making use of exist techniques to transform them into our research domain, in order to obtain an expected enhancement or impel the related researches.

Towards this procedure, most important sticking point is to find an efficient mechanism to query and inference the description data so as to get high performance look-up and provide semantic level matching rather than syntax level. Actually, there are two approaches to the problems: one is the data-mining; the other is logical retrieval methods. In this thesis, we concentrate mainly on the latter.

3.2 Making Use of Existing Technologies

One major difference between the approach taken in this thesis and other projects is the requirement to use existing computing technologies whenever possible instead of developing proprietary solutions. It is the opinion of the author that dynamic adaptation techniques will not be embraced and widely deployed if they require software which is too specialized or complex. A major contribution that this thesis provides is to design an architecture that makes use of the available software and carries out runtime discovery of services and components.

3.2.1 Ontology Engineering

Ontology engineering, as proposed in e.g. [10], is a research methodology which gives us design rationale of a knowledge base, kernel conceptualization of the world of interest, strict definition of basic meanings of basic concepts together with sophisticated theories and

technologies enabling accumulation of knowledge which is dispensable for modeling the real world.

The body of knowledge is based on a conceptualization: the objects, concepts, and other entities that are assumed to exist in some area of interest and the relationship that hold among them. A conceptualization is an abstract, simplified view of the world that we wish to represent for some purpose. When the knowledge of a domain is represented in a declarative formalism, the set objects that can be represented is called the universe of discourse. This set of objects and the describable relationships among them are reflected in the representational term. In such an ontology, definitions associate the names of entities in the universe of discourse (e.g. classes, relations, functions, or other objects) with human readable text describing what the names mean, and formal axioms that constrain the interpretation and well-formed use of these terms, Formally it can be said that an ontology is a statement of a logical theory [11].

In this sense, ontology is important for the purpose of enabling knowledge sharing and reuse, and while these two aspects are critical to the semantic discovery. Because if we create a proper and powerful ontology for services taxonomy, it will sure for different components to exchange information using uniform description languages. That will facilitate recognizing and interacting among the two procedures mentioned before.

3.2.2 RDF/S in a Nutshell

The Resources Description Framework and Schema Language (RDF/S) [12][13] aim to facilitate the encoding, exchange, processing and reuse of resource meta-data while each user community is free to specify its own description semantics in a standardized, interoperable, human-readable manner via and XML-based infrastructure [14].

The RDF data model is based on the notion of "resource", everything, concept or object, available on the web or not, can be modeled as a resource identified by a unique URI [15] [33]. With the constructs of the RDF data model we describe interrelationships among resources in terms of named properties and values. Properties capture either attributes of a resource or binary relationships between resources. The definition of these attributes/relationships and their semantical attribution is accomplished trough the RDF Schema Language (RDFS) [13].

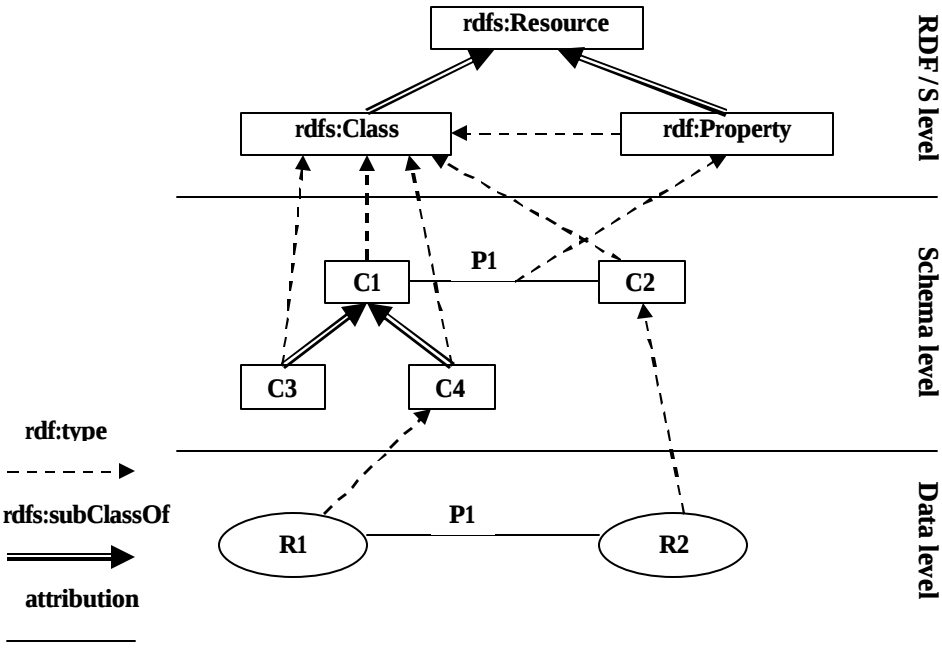


Fig. 2 Abstraction levels in RDF/S Schema

An RDF Schema declaration is expressed in the basic RDF Model and Syntax Specification and consists of classes and properties. In other words, the RDF Schema mechanism provides a type system for RDF models, i.e., a vocabulary of the valid terms that can be used to describe resources. A brief summary of the basic RDF/S features are drawn as follows:

Core class: The basic constructs of the RDF/S meta-language are *Class*, *Property* and *Container*, which correspond to entities, relations or attributes and complex or structured values, respectively.

Abstraction mechanism: RDF/S features the following abstraction, mechanisms: (multiple) class or property inheritance and (multiple) classification of resources. The former is declared using the *rdfs:subClassOf* or *rdfs:subPropertyOf* core properties while the latter using the *rdf:type* core property. Typically, we identify three core abstraction levels, which are depicted in Figure 2.

Restriction mechanisms: Although RDF/S does not provide elaborate mechanisms for defining property restrictions (as in the case of Description Logic or frame languages), we can declare simple domain and range restrictions through the *rdfs:domain* and *rdfs:range* core properties.

Documentation facilities: The properties *label*, *comment*, *isDefinedBy* and *seeAlso* are used to document the development of a schema.

Reification mechanism: Although not expressible at schema level, RDF provides mechanisms for representing statements. This mechanism, formally known as reification, is applicable at the data level and the constructs used for this process are *statement*, *subject*, *predicate*, *object* and *type*.

The XML namespace facility [16] plays a crucial role in the development of RDF schemas, since it enables the reuse of terms from other schemas. With the use of an XML namespace, descriptive terms (i.e., class or property names) are uniquely identified by URI (i.e., play the role of a name prefix) as normal web resources.

3.2.3 Schema Definition Concepts

Since in our solution, the most important mechanism for dynamic adaptation is by means of using the schema concepts provided by RDFS language. So a brief introduction for the core vocabularies, it is also the basis for the following mapping from RDF to Conceptual Graph.

All those concepts are defined in [19], the second document of W3C, to allow the definition of schemas, vocabularies of resources to use with RDF. In schemas, new resources can be defined as specialization of old ones, thus allowing inferring implicit triples. Schemas also constrain the context in which defined resources may be used, inducing the notion of schema validity. They all can be expressed as rules allowing inferring new facts (basically, new triples or negations of triples). In these rules, the 3-ary logical predicate T(*subject*, *predicate*, *object*) will be used to represent a believed triple.

3.2.3.1 rdfs:subPropertyOf

Any property denotes a relation between resources (the set of resource couples linked by an arc labeled with the property). *rdfs:subPropertyOf* applies to properties and must be interpreted as the subset relation between the relations they denote. Thus the following rule stands:

$$\forall s, p_1, o, p_2 \; \Gamma(s, p_1, o) \wedge \Gamma(p_1, \text{rdfs} : \text{subPropertyOf}, p_2) \Rightarrow \Gamma(s, p_2, o)$$

For example, if "mother:" is a sub-property of "parent", any triple having "mother" as predicate must also be considered as having "parent" as predicate. This property is very important in schema definitions for interoperability between RDF agents. In the example above, an agent not knowing the semantics of "mother" could at least treat it as "parent" (assuming it knows the semantics of "parent").

Since *rdfs:subPropertyOf* denotes a subset relation, the transitivity rule also stands:

$$\forall p_1, p_2, p_3$$

$$\Gamma(p_1, \text{rdf} : \text{subPropertyOf}, p_2) \wedge \Gamma(p_2, \text{rdfs} : \text{subpropertyOf}, p_3)$$

$$\Rightarrow \Gamma(p_1, \text{rdf} : \text{subpropertyOf}, p_3)$$

$$\textbf{and} \;\; \forall p \neg\Gamma(p, \text{rdfs} : \text{subpropertyOf}, p)$$

It is considered invalid to have cycles in *rdfs:subPropertyOf*, though it doesn't define a way to express this constrain in RDF.

3.2.3.2 rdf:Class, rdf:type and rdfs:subClassOf

Classes are resources denoting a set of resources, by the mean of the property *rdf:type* (instances have property *rdf:type* valued by the class). Since all sets of resources presented in this section are resources (each has a URI), they have by definition the property *rdf:type* valued by *rdfs:Class*. On the other hand, all properties have *rdf:type* valued by *rdf:Property*.

Classes are structured the same way as properties, in a subset hierarchy denoted by the property *rdfs:subClassOf*. As for *rdfs:subPropertyOf*, cycles must not exist though it could be used to express equivalence, but contrary to the property hierarchy, the class hierarchy has a maximum element: it is of course *rdf:Resource* (so any class implicitly has *rdfs:subClassOf* valued by *rdf:Resource*). The following rules, similar to the rules related to *rdfs:subPropertyOf*, stand:

$$\forall i, c_1, c_2 \; \Gamma(r, \text{rdf} : \text{type}, c_1) \wedge \Gamma(c_1, \text{rdfs} : \text{subClassOf}, c_2)$$

$$\Rightarrow \Gamma(i, \text{rdf} : \text{type}, c_2)$$

$$\forall c_1, c_2, c_3 \; \Gamma(c_1, \text{rdfs} : \text{subClassOf}, c_2) \wedge \Gamma(c_2, \text{rdfs} : \text{subClassOf}, c_3)$$

$$\Rightarrow \Gamma(c_1, \text{rdfs} : \text{subClassOf}, c_3)$$

$$\forall c \; \neg\Gamma(c, \text{rdfs} : \text{subClassOf}, c)$$

3.2.3.3 rdfs:domain and rdfs:range

These properties apply to properties and must be valued by classes. They are used to restrict the set of resources that have a given property (the property's domain) and the set of valid values for a property (its range). A property may have as many values for *rdfs:domain* as needed, but no more than one value for *rdfs:range*:

$$\forall p_1, r_1, r_2 \; \Gamma(p, \text{rdfs} : \text{range}, r_2) \wedge r_1 \neq r_2 \Rightarrow \neg\Gamma(p, \text{rdfs} : \text{range}, r_2)$$

For a triple to be valid, the object must match the range (if any) of the predicate (that is, it must have *rdf:type* valued by the corresponding class or one of its subclasses), and the subject must match at least one of the domains (if any) of the predicate that is, it must have *rdf:type* valued by the corresponding class or one of its subclasses), and the subject must match at least one of the domains (if any) of the predicate. This can be logically expressed by:

$$\forall s, p, o \; \Gamma(s, p, o) \wedge \exists d \; \Gamma(p, \text{rdfs} : \text{domain}, d)$$

$$\exists d'(\Gamma(p, \text{rdfs} : \text{domain}, d') \wedge \Gamma(s, \text{rdf} : \text{type}, d'))$$

$$\forall s, p, o, r\ \Gamma(s, p, o) \wedge \Gamma(p, \text{rdfs} : \text{range}, r) \Rightarrow \Gamma(o, \text{rdf} : \text{type}, r)$$

It is worth nothing that, although this two rules are intended to be used for validity checking only, and the first one (*rdfs:domain*) can actually only be used this way (it can not be used to perform inference since its consequence is existentially qualified), the second one (*rdfs:range*) has different interpretation depending on hypothesizing a closed or open world. In the closed world hypothesis, any missing triple is considered negated, so the *rdfs:range* rule has only to be verified. But in an open world hypothesis, missing triples are not necessarily false, so the rule could be used to perform inference instead. Since the field of RDF that we focus on is the services description, during the discovery the information is essence distributed and sometimes incomplete, the open world hypothesis seems much more reasonable.

3.2.3.4 Example

After we present the foundation RDF/S syntax and relationship among the resources in the RDFS hierarchy, here a brief example would help to gain a more concrete view for the previous introduction. The example is extracted from the RDF files which the basic source for the future discovery system to be designed. First is the system schema used to creating the service ontology to give the criteria to describe a certain service component.

```
<?xml version="1.0" ?>
<rdf:RDF
xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs  = "http://www.w3.org/2000/01/rdf-schema#"
xmlns:xs = "http://www.w3.org/1999/XMLSchema-datatypes#">

<rdfs:Class rdf:ID="Component">
<rdfs:label>Component</rdfs:label>
<rdfs:comment></rdfs:comment>
</rdfs:Class>

<rdf:Property rdf:ID="Description">
<rdfs:domain rdf:resource="#Component"/>
<rdfs:range rdf:resource="#DescriptionClass"/>
</rdf:Property>

<rdfs:Class rdf:ID="DescriptionClass"/>
```

```
<rdf:Property rdf:ID="ServiceName">
<rdfs:domain rdf:resource="#DescriptionClass"/>
<rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>

<rdf:Property rdf:ID="ServiceAlias">
<rdfs:domain rdf:resource="#DescriptionClass" />
<rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>

<rdf:Property rdf:ID="ClientName">
<rdfs:domain rdf:resource="#DescriptionClass"/>
<rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>

<rdf:Property rdf:ID="Capability">
<rdfs:domain rdf:resource="#DescriptionClass"/>
<rdfs:range rdf:resource="#CapabilityClass"/>
</rdf:Property>

<rdf:Property rdf:ID="Requirements">
<rdfs:domain rdf:resource="#DescriptionClass"/>
<rdfs:range rdf:resource="#RequirementsClass"/>
</rdf:Property>
... ...
... ...
</rdf:RDF>
```

In this RDF schema we have defined many of the classes and related properties, and definitude their relationship to establish a primary service ontology. Thus they can be used to standardize the description for service component, and the concrete RDF descriptions are stipulated in the data level.

```
<rdf:RDF xml:lang="en"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:s="http://hostname/2002/services.rdfs#">

<s:Component rdf:about="http://hostname/2002/compaq/">
<s:Description>
<s:DescriptionClass>
<s:ServiceName>Compaq Ipaq</s:ServiceName>
<s:ServiceAlias>3800 series</s:ServiceAlias>
<s:ClientName>EMOOSE 2002</s:ClientName>
<s:Capability>
<s:CapabilityClass>
```

<s:ClientCapability>1000</s:ClientCapability>
</s:CapabilityClass>
</s:Capability>
<s:Requirements>
<s:RequirementsClass>
<s:CPURequirement>33MHz</s:CPURequirement>
<s:MemoryRequirement>16M</s:MemoryRequirement>
<s:DiskRequirement>500M</s:DiskRequirement>
</s:RequirementsClass>
</s:Requirements>
</s:DescriptionClass>
</s:Description>
</s:Component>

</rdf:RDF>

This example would be further used in the next chapter when discussing the design of architecture. And we would see how to handle the discrepancy when the component uses other descriptive resources rather than the one it should obey to use.

3.2.4 Why Choose RDF/S

Above, we briefly present the two technologies: ontology engineering and RDF/S language. Actually, there are not only one description language used in ontology, for example, OIL and DAML. So in the followings we would elucidate why we choose RDF/S as our basic solution tools, and also see its limitations which would be the part of task to conquer in this thesis. Moreover, the advantages made of by using ontology in service discovery are showed in the context.

As an ontology language, there are six design goals for it to support [34]. The task and rationale are explained in turns.

3.2.4.1 Shared Ontologies

Ontologies should be publicly available and different data sources should be able to commit to the same ontology for share meaning. Also, ontologies should be able to extend other ontologies in order to provide additional definitions.

The key feature of semantic discovery is interoperability. It is obvious that interoperability requires agreements on the definitions of terms. Ontologies can provide standard sets of

terms and formal descriptions of those terms. Data sources that commit to the same ontology explicitly agree to use the same terms with the same meanings.

In RDF, each schema has its own namespace identified by URI. Each term in the schema is identified by coming the schema's URI with term's ID. Any resource that uses this URI references the term as defined in that schema. By this means it can realizes interoperability in some degree.

3.2.4.2 Ontology Evolution

Ontologies can be changed over time and data sources should specify which version of the ontology they commit to. Any use case in which the ontology could potentially change, and in particular those in which the owner of the ontology is different from the data providers.

Since the services whichever are software or hardware are constantly growing and changing, we must expect ontologies change as well. Ontologies may need to change because they were errors in prior versions, because a new way of modeling the domain is preferred, or because reality has changed (e.g., the addition of new technology). A Web ontology language must be able to accommodate ontology revision. Note that ontology evolution is different from ontology extension, which does not change the original ontology. An important issue of revision is whether or not documents that commit to one version of an ontology are compatible with those that commit to another. Both compatible revisions are allowed. It is possible for a revision to change the intended meaning of a term without changing its formal description. Thus determining semantic backwards-compatibility requires more than a simple comparison of term descriptions. As such, the ontology author needs to be able to indicate such changes explicitly. So a good mechanism to support ontology evolution is critical to our work. In other word, what we mention before, self-update and self-mend are the feature the semantic architecture should have.

In RDF Schema the *rdf:subClassOf* and *rdfs:subpropertyOf* properties can be used to relate new versions of classes and properties to older versions. However, sometimes this has the drawback that incorrect definitions cannot be retracted.

3.2.4.3 Ontology Interoperability

Different ontologies may model the same concepts in different ways. The Language should not provide primitives for relating different representations, thus allowing data to be converted to different ontologies and enabling distinct descriptions can be comprehended cross-platform.

Although shared ontologies and ontology extension allow a certain degree of interoperability between different organizations and domain, there are often cases where there are multiple ways to model the same information. Any use case in which data from different providers with different terminologies must be integrated. RDF provides minimal support for interoperability by means of the *rdfs:subClassOf* and *rdfs:subpropertyOf* properties. These are not efficient, and more contribution would be made for this goal in the thesis later.

3.2.4.4 Balance of Expressivity and Scalability

The language should be able to express a wide variety of knowledge, but should also provide for efficient means to reason with it. Since these two requirements typically at odds, the goal of the ontology language is to find a balance that supports the ability to express the most important kinds of knowledge. And this knowledge here is the knowledge of services and components description and internal relations that would be used in the process of discovery.

The diversity of the services is beyond common imagine, and meantime the potential application of the software or hardware components to embedded devices and agents poses even larger amounts of information that must be handled. The ontology language should support reasoning systems that scale well. However, the language should also be as expressive as possible, so that users can state the kinds of knowledge that is important to their applications.

Expressivity determines what can be said in the language, and thus determines its inferential power and what reasoning capabilities should be expected in systems that fully implement it. An expressive language contains a rich set of primitives that allow a wide variety of knowledge to be formalized. A language with too little expressivity will provide too few reasoning opportunities to mush use and may not provide any contribution over existing

languages. RDF is very scalable (with the exception of XML syntax being extremely verbose) but is not very expressive.

3.2.4.5 XML Syntax

It is clear that exchange of ontologies and data in a standard format is of consequence in the design of an ontology description language. XML has become widely accepted by industry and numerous tools or processing XML have been developed. If an ontology language has XML syntax, like RDF, then theses tools can be extended and reused.

3.2.4.6 Easy Use

The language should provide a low learning barrier and have clear concepts and meaning. Since the data documents are marked up and queried by user, and though ideally most users will be isolated from the language by front end tools, the basic philosophy of the language must be natural and easy to learn. Further, early adapters, tool developers, and power users may work directly with the syntax, meaning human readable (and writable) syntax is desirable.

RDF/S is earlier to learn with the back ground of XML syntax, and it is between the machine readable language and human readable language.

Above, we illustrate what are the design goals for ontology language. Together, we show the reason why to choose RDF/S as the basic developing ontology language. In faith, the limitations of RDF in the way to design a powerful ontology model are actually our task to makeup, and more important is we transform it into the field of semantic services discovery and endue it with new application area.

3.2.5 Concept Graph

As RDF is selected as our basic ontology language, which the architecture infrastructure would be founded on, still there are several problems left, such as: searching information (i.e., match keyword), extracting information, generate documents. We are convinced of the interest of Artificial Intelligence representation languages that enable not only the representation of metadata but also support inferences on them. Among such AI knowledge representation formalisms, [17] [18] stress the advantages of conceptual graph (CG) formalism for expressing meta-data. Another approach is to exploit a standard language for

expressing meta-data in conceptual graphs in order to exploit querying and inference capabilities enabled by conceptual graph formalism.

The specification of Conceptual Graph Architecture is beyond the subject in this thesis. As CG is a logical reasoning tool like Prolog, we would not concentrate on the explanation of how it is, but the way how to transform it into our architecture and take advantage of its logical reasoning abilities. However, a few sections of overview on CG are written just to show the idea, and how to integrate such a tool in the architecture would be set forth in the chapter on design.

Conceptual Graph (CG) is a system of logic based on the existential graphs of the semantic networks of artificial intelligence. The purpose is to express meaning in a form that is logically precise, humanly readable, and computationally tractable. With the direct mapping to language, conceptual graphs can serve as an intermediate language for translating computer-oriented formalisms to and from natural languages. With their graphic representation, they can serve as a readable, but formal design and specification language. CG has been implemented in a variety of projects for information retrieval, database design, expert systems, and natural language processing.

Conceptual Graph is formally defined by an abstract syntax that is independent of any notation, but the formalism can be represented in either graphical or character-based notations. Informally, a CG is a structure of concepts and conceptual relations where every arc links a concept node and a conceptual relation node. Formally, the abstract syntax specifies conceptual graph as mathematical structures without making any commitments to any concrete notation or implementation. CG may be implemented in any machine-readable representation or any humanly readable style that preserves the information specified by the abstract syntax.

To illustrate the abstract syntax and concrete notations, Figure 3 show the display form of a conceptual graph that represents the propositional content of the English sentence *John is going to Paris by train*.

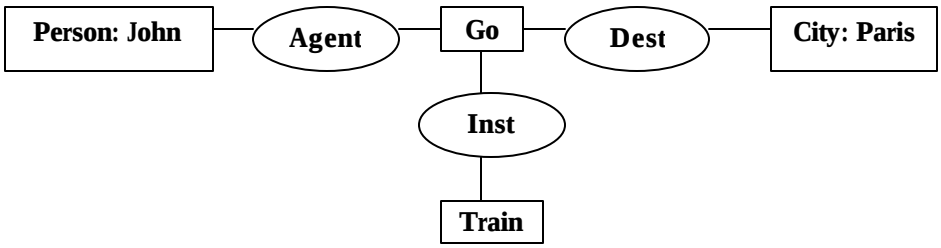


Fig. 3 CG Display Form for "John is going to Paris by train."

In DF, concepts are represented by rectangles: [Go], [Person: John], [City: Paris], and [Train]. Conceptual relations are represented by circles or ovals: (Agnt) relates [Go] to the agent John, (Dest) relates [Go] to the destination Paris, and (Inst) relates [Go] to the instrument bus. For n-adic relations, the arcs are numbered from 1 to n; for dyadic relations, the number 1 may be replaced by an arrowhead pointing toward the relation, and the number 2 may be replaced by an arrowhead point away from the relation.

As a mnemonic aid, an arrow pointing toward the circle may be read "has a(n)"; an arrow pointing away may be read "which is a(n)"; and any abbreviations may be expanded to the full forms. With this convention, Figure 3 could be read as three English sentences:

- ✧ Go has an agent which is a person John.
- ✧ Go has a destination which is a city Paris.
- ✧ Go has an instrument which is a train.

This English reading is a convenience that has no normative status. The numbering or direction of the arcs takes precedence over any such mnemonics.

The linear from for CG is intended as a more compact notation than DF, but with good human readability. It is exactly equivalent in expressive power to the abstract syntax and the display form. Following is the Logical Form for Figure 3:

[Go] –

(Agnt)->[Person: John]

(Dest)->[City: Paris]

(Inst)->[Train].

In this form, the concepts are represented by square brackets instead of boxes, and the conceptual relations are represented by parentheses instead of circles. A hyphen at the end of a line indicates that the relations attached to the concept are continued on subsequent lines.

Both Display Form (DF) and Logical Form are designed for communication with humans or between humans and machines. For communication between machines, the Conceptual Graph Interchange Form (CGIF) has a simpler syntax and a more restricted character set. Following is the CGIF for Figure 3:

[Go *x] (Agnt ?x [Person: John]) (Dest ?x [City: Paris]) (Inst ?x [Train])

CGIF is intended for transfer between IT systems that use CG as their internal representation. For communication with systems that use other internal representations, CGIF can be translated to Knowledge Interchange Format (KIF):

(exist ((?x Go) (?y Person) (?z City) (?w Train))

(and (Name ?y John) (Name ?z Paris)

(Agnt ?x ?y) (Dest ?x ?z) (Inst ?x ?w)))

Although DF, LF, CGIF, and KIF look very different, their semantics is defined by the same logical foundations. Any semantic information expressed in any one of them can be translated to the others without loss or distortion. Formatting and stylistic information, however, may be lost in translations between DF, LF, CGIF, and KIF.

3.2.6 Mapping RDF to CG

The model of CG formalism [19][20] is based on a support made of a concept type lattice and of a relation type set possibly organized in hierarchy, a set of individual markers enabling the designation of instances, a conformity relation between markers and types, and a base of conceptual graphs built on this support.

It therefore seems natural to translate the RDF statements into a base of CG-facts; the hierarchy of classes appearing in a RDF schema into a concept type hierarchy in CG, and the hierarchy of properties appearing in a RDF schema into a relation type hierarchy in CG. Therefore we will rely on a CG model enabling us to build a relation type hierarchy.

In [20] Oliver Corby offers a good and doable approach to realize the mapping work. In thesis we would use the reference form his model, and more convenient is that in his project Corese: A Conceptual Resource Search Engine it provides an elementary API to make use of the mapping model. So we would integrate this module into our architecture in order to get a performance of reasoning and inferencing to help discovery process by means of logical conceptual methods [39].

3.2.6.1 Mapping RDF Schema

RDFS classes can be modeled as CG concept types. In order to map Resource core class of RDFS, we introduce a Resource concept type at the top level of the CG concept type hierarchy.

concept type resource

A RDFS class without any super class explicitly indicated will be modeled by a subtype of Resource in the CG concept type hierarchy.

<rdfs:Class rdf:ID = Cl '/>

This can be translated into,

Concept type Cl < Resource

For example:

<rdfs:Class rdf:ID = Book ’/>

can be modeled as:

concept type Book < Resource

3.2.6.2 Mapping Subclasses

The subClassOf relation between classes in RDFS corresponds to the subtype relation between concept types in CG formalism. If an RDFS class Cl2 is defined as a subclass of Cl1, it will be modeled by a Cl2 concept type, subtype of the Cl1 concept type in the CG concept type hierarchy.

<rdfs:Class rdf:ID = Cl2 ’>
<rdfs:subClassof rdf:resource = #Cl1 ’/>
</rdfs:Class>

This can be translated into:

concpet type Cl2 < Cl1

The Novel subclass of Book:

<rdfs:Class rdf:ID = Novel ’>
<rdfs:subClassOf rdf:resource = #Book ’/>
</rdfs:Class>

can be modeled as subtype of Book:

concept type Novel < Book

3.2.6.3 Mapping of Properties

A property is defined according to a domain (i.e. a class) and has an associated range that can be a literal or a class. For example, the title property can be defined with Book ’domain and Literal ’range. Literal ’is prefixed with the RDFS namespace.

<rdf:propertt ID = title ’>

<rdfs:domain rdf:resource = "#Book "/>
<rdfs:range rdf:resource = "http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Literal"/>
</rdf:Property>

A property definition can be modeled as a CG binary relation type with an associated signature that maps the domain and range to the related concept types. For instance, the previous example is translated into:

relation type title (Book, Literal)

3.2.6.4 Mapping of Subproperties

In RDFS, a subproperty can refine an existing property, in the same way as a subclass refines a class. It means that if p2 is a subproperty of p1, then:

p2(URI, v) => p1(URI, v)

that is, if a URI has value v for property p2, then it salso has v as value for property p1. For example, define the author property and its jointAuthor subproperty:

<rdf:Property ID = "author">
<rdfs:domain rdf:resourse = "#Book "/>
<rdfs:range rdf:resource = "http://www.w3.org/TR/1999/PR-rdf-schema-19990303#Literal"/>
</rdf:Property>
<rdf:Property ID = "jointAuthor">
<rdfs:subPropertyOf rdf:resource = "#author "/>
</rdf:Property>

In terms of CG, an RDF subproperty is translated into a relation type that a subtype of the relation type translating the superproperty. In the example above, the jointAuthor relation type will be defined as a subtype of the author relation type.

relation type author (Book, Literal)

relation type jointAuthor < author

Above, we briefly present the way how RDF schema can be translated into Conceptual Graph, and by using this mapping engine, it will contribute to the reasoning process for

semantic discovery among the services and components which describe its ontology by RDF/S language.

3.3 .NET Handheld Apps Develop

As in the final implementation, we need to implement an application to carry out a practical ubiquitous computing scenario. Microsoft .NET provides rich support for handheld application development and the recently it releases a new generation tools for smart devices application under .NET, which is called .NET Compact Framework. Due to the high efficiency and compatible with XML based technology which is an ambitious strategy of Microsoft, the client side is developed under Smart Device Extension, a plug in for Visual Studio, and utilizing Web Service to be a shell outside the core classes, which interact with the user and transmit the query and response. The detail structure would be present in the next chapter, and in this section a brief overview on development of .NET handheld application is made to explain its relationship with towards architecture.

Smart Device Extensions (SDE) for Visual Studio .NET allow programmers to develop applications for the NET Compact Framework, a new platform that maintains many of the features of the .NET Framework in a version optimized for handheld devices. In developing applications with SDE, the developers are targeting the .NET Compact Framework, which simplifies application development on smart devices.

The .NET Compact Framework was designed to be easily ported to other platforms, whether those platforms were created by Microsoft or third-party vendors. So the application created on a Pocket PC could just easily run on other platforms such as a cell phone or other vendor's PDA, provided that a version of the .NET Compact Framework has been implemented for that platform.

The classes in the .NET Compact Framework have an interface identical to their .NET Framework equivalents with the exception of functionality that is not supported because of size constraints, performance issues, or limitations in the target operating system. Class behaviors, properties, methods, and enumeration values are the same under both versions of the .NET Framework.

The compact version of the .NET Framework implements a subset of the System.Windows.Froms and System.Drawing classes. These classes can be used to construct rich, Windows CE-based user interfaces for device applications. Much of the interaction with these classes is managed for the developers by the Windows Forms designer component of Visual Studio.NET. Implementation of Windows Forms under the .NET Compact Framework includes support for forms, bitmaps and menus, most controls in the .NET Framework, and the ability to host third-party controls.

The .NET Compact Framework also provides a subset of the Web Services functionality offered in the .NET Framework. Most significantly, using VS and SDE applications can be created that allow easily consuming XML Web Services. So it is very reasonable to create Web Services and client applications that consume Web Services using Visual Studio .NET.

The .NET compact Framework also provides support for the basic Graphical Device Interface (GDI) drawing elements including bitmaps, brushes, fonts, icons, and pens. Moreover, the .NET Compact Framework provides a set of base classes that expose a wide range of functionality. This underlying infrastructure enables you to write applications for .NET that incorporate multithreading, take advantage network resources.

Finally, what makes it convenient is that kinds of images can run in the emulator act like the actual devices. Applications running in the emulator can access the network, consume Web Services, install software, and perform any other action. Not only does the emulator included with SDE provide as accurate representation of the actual device, it also allows modify, save, and store custom configurations of operating system images.

3.4 Strategy for the Thesis

After examining the several foundation techniques in the previous sections, the holistic strategy for the architecture is to be present here. To make the fragmentary techniques all together and better integrate into the infrastructure and take advantages from their strongpoint, it is critical to illuminate the direction before stepping into the design part. And moreover, this section is also thought as a summary of research focus for this chapter.

Having surveyed the existing techniques which can be integrated into the architecture, the main work can be generalized into three steps:

1. Establish a dynamic ontology by means of the ontology description language RDF/S, which provides a solid ontology infrastructure, and carry out a high semantic-level for services and components description.

2. Constitute an internal reasoning and inferencing system which based on Conceptual Graph, together with the mapping mechanism between RDF and CG. This part should offer a semantic discovery and query functions in some degree. And it should also seamless lined with the formal part.

3. Implement a handheld application under .NET compact Framework, which interacts with the end point users, accepts the querying and communicates with the services registry. To program the application consuming the discovery function, a common client/server model is to be adopted. Though this part is not the key point of this thesis, a good user interface is still of importance for the demonstration.

One thing should be stated is that since currently most of the RDF/S parsers are developed under Java programming, so in the first two parts Java would be the implementation language. Though the final and ideal envisage is to construct the whole infrastructure in the Windows .NET platform, in fact the core methodology and main contribution are in the meta-data level, and the present prototype could be wholly transformed and reconstituted in the .NET environment. Moreover, the ontology languages supported by Java are more mature than those in the .NET, for .NET is totally a new born idea who is lack of followers in ontology domain during the current period. So it could the future work to be discussed in the last chapter.

Clear with the blueprint and each facet of the requirements, the detail design of architecture is to be present in the next chapter.

Chapter 4 Design of Architecture

4.1 Overview

This chapter describes a design for general purpose semantic service discovery architecture. The architecture will provide the entire required infrastructure to construct a semantic service adaptation and discovery system that servers for not only ubiquitous computing device 'users to lookup the services they need, but also the service programmers to advertise their own components to integrate with the searching schema. The criteria fro a service component will be outlined later in the chapter. There are three primary techniques that are supported in the architecture. While these techniques have been described previously in the last chapter, brief outline will be summarized next.

The first technique allows the description of a service component. All of the processes of the architecting are based on the way that we describe the various services components. As we selected before, to create the rudimental ontology to support further dynamic and discoverable system, Resource Description Framework is the language we used to construct the ontology schema and depict the components. It is a two-level structure in the development of component data: one is using RDF language to describe the common service; the other is the RDF Schema which organizes the meta-data structure to explain the way to describe the service and the creation of the semantic ontologies. The design of these two aspects would be detailed in this chapter later.

The second technique is Conceptual Graph to server as the discovery part and offering inferencing and reasoning function in the architecture. As it is mentioned before, CG is a good approach to the logical data searches and able to realize a high level performance in the retrieval of ontologies data. Moreover, a bridge can be made that mapped the RDF described ontology into the Conceptual Graph formalism. Some existing infrastructure, like Corese, offers Java API to facilitate the mapping work and can be integrated into our project to play an important role to link the dynamic ontology part with service discovery part.

The third one, .NET handheld application development, is for the implement of a feasible ubiquitous computing scenario. And with the help of the convenient developing tools and

technology, such as Smart Device Extension and Web Service, the idea realized in the formal two parts would be easily achieved in the ubiquitous distributed devices environment. Further, the final assumption is to convert the whole prototype into the .NET platform, which means not only to make use of the .NET handheld infrastructure for client development, but also turn the adaptation and retrieval part into the .NET environment, so as to keep the consistency and take great advantage from the prolific platform resources.

Before we describe the requirements and design of the architecture, a few general assumptions need to define in advance. These assumptions allow limiting the architecture and eventual implementation to the specific challenges of ontologies and semantic discovery in this thesis. And these assumptions will attempt to justify the design decisions throughout this chapter.

4.2 General Assumptions

One of the central objectives in this thesis is to determine how to make a component describable ontology that can dynamically adapt to the various services depicted in the same way. That means whatever various the services components could be, in order to make them integrated they must basically be specified in the uniform description methods. For this reason, one assumption is, although different services have different properties and features, when they are about to involve in the discovery system and advertise in the services repository, each of them must be specified in the accordant language. In this thesis as we state before, RDF/S language is used to be ontology description language. So the system has an initial and basic meta-data schema which formalizes the way to construct the RDF data. The discrepancy lies on each of the services may partly change the semantization in the ontology, but this does not mean they would convert the entireness into another way. The constraints of the changes result in the second assumption.

The second assumption could be thought as an important limitation that has been anticipated in the design architecture. That is, there are some constrains on the particular service that may result the system ontology schema need to re-mend and update in order to adapt the new alters have taken place. These new alters would be limited like add new property, add new class and change the range and domain and so on. They would be specified at length in the following section. To state the constrain and limitation of the services description is because

the goal of adaptation has its own capacity to deal with the diversity. The adaptation do not means that every kind of the alters can be handled, and it only make sense when the new changes would contribute to enrich the scope of the ontology schema.

Finally, since the most essential thing is to find a good expressive ontology, while not only to focus on how the service works. All the resulting services will be functional but not necessarily efficient or useful, which means the services components provided for test could be not really exist or usable. But they are actually modeled from the practical components, and more representative so as to be able to stand for the real ones.

4.3 Requirement Analysis

This section will outline the requirements for the dynamic ontology and service discovery infrastructure. However, before requirements can be captured, the environment the system will operate in that the functionality provided by the system itself must be identified.

4.3.1 Identifying System Roles

The entities that make use of the system and affect how the system operates can be thought of possessing the unique roles in the system. Since it is not a complex system, only three main actors have been identified, namely, the client, the server, the service provider.

The client interacts with the system to request a specific service. It is presented with a user interface by the system which allows the query to be sent. And the services to be searched are based on a set of attributes selected and provided by the client. The client in this context could be a ubiquitous computing device, or a distributed computer.

The server receives the requests and performs locating and retrieving a service component which match the criteria provided by a client. It not only interacts with the client, but also accepts the new coming services to register in its repository, moreover, acclimatizes itself to enrich the descriptive capacity, so as to server better for the service discovery.

The service provider is responsible for creating service components, registering them with the system, and uploading the description to the system's service component repository when it

need to be retrieve. The system uses the related information to constitute the ontology and executive the discovery process.

4.3.2 List of System Requirements

The particular requirement can be defined after the identification of the system roles. As we have already presented the main requirement in the first chapter as introductions, the following items are just to detail the sub work and define the tasks according to the roles.

- ✧ Facilities to create an internal ontology that has the capacity to describe the components, the features, the attributes and the relationships
- ✧ Facilities to allow ontology schema to update duly so as to enrich the knowledge base
- ✧ Facilities to allow an entity to locate and discover the services components
- ✧ Facilities for service component lookup and retrieval
- ✧ Facilities for the new coming service component enrolled in the service repository
- ✧ Facilities for analyzing service component specification
- ✧ A mechanism to conquer the discrepancy when mend the schema
- ✧ A mechanism to map the RDF data into the CG formalism so as to realize the logical retrieval
- ✧ Facilities for handheld devices to communicate with the discovery service in the distributed internet environment
- ✧ Facilities to access the discovery service and easily to construct the query message by offing good user interface on the client end

4.3.3 Requirement Modeling

As the roles and responsibility have been assigned, and together with two procedures which we have already illustrate in chapter 3, that is, adaptation procedure and discovery procedure, the Unified Modeling Language (UML) can be used to illustrate the relationship with each roles and how they interact with the system. In particular, a context diagram is shown below.

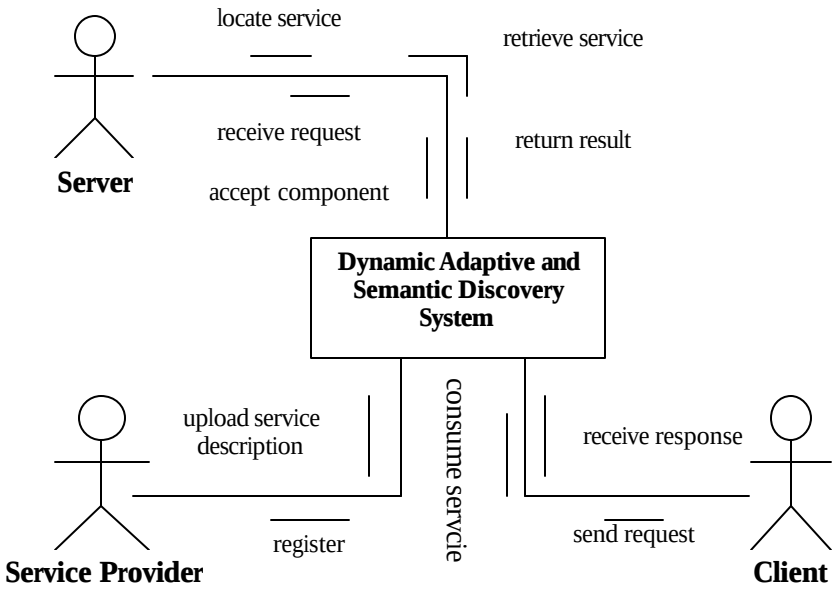


Fig. 4 Context Diagram for System

4.4 Operational and Functional Description

After the previous high-level view on the system, this section will describe the system's functional characters in detail. To make it clear, two scenarios of how to accomplish the real cases will be presented together with the descriptions, so that the requirements and design issues can be uncovered.

4.4.1 Generic Ontology Construction

Before the discovery procedure can be carried out, a solid knowledge base should be constructed in advance so as to have the capacity to reason and reference from the pre-known information. This is very critical to the latter process and could be thought of as the

person's knowledge accumulation process, that is, one cannot know what is true or false before he has read enough to select and determine.

First we construct the fundamental ontology schema which has a basic ability to direct the component description. For example, as a common service, each service component possess service name, service capacity description, service requirement description, service input/output description and so on. After we widely and precisely survey on the commonness of current existing ubiquitous services, a skeleton ontology schema has been drawn to be the fundament of service descriptions. To thought it as a human skeleton is very visualizing, because though different services may have their own characters and need some special way or syntax to describe, most of the services components indeed share a lot of features with each other. So the characteristic could be considered as the body constituted upon the skeleton, and then result the diversity among the collectivity.

The concrete schema is modeled according to the RDFS syntax so as to take the advantage of the ontology language to have the powerful capacity to deal with the complex relationship among the items. The whole schema is shown as Fig. 5.

In the schema, we could see the entity of a service component is made of a basic class named as *Component*, which is the subclass of Resource in the RDF hierarchy. It has two properties, *Description* and *Property*, which using *DescriptionClass* and *PropertyClass* to describe themselves. All of the other features are derived from these two classes, so that more specified and particular facets which are to describe the service more precisely are involved and even could be reviewed and mended if needed. These means the initial schema may be partly different from the one after its update now and then, which is similar to the evolution procedure to grow and meliorate when more and more new services come and enroll in this system to participate in the service discovery process.

Initially, the schema describes the services from description and property these two aspects, the *DescriptionClass* mainly focuses on the depiction of the common characters. For example, it has *ServiceName*, *ServiceAlias* to distinguish a particular component, and *Capacity*, *Cost* to show the volumes and consumes, and *Input*, *Output* properties to present the type of information used to be exchanged between server and client. The *RequirmentClass* is design to explain the

particular environment needed by the service to run in. Even more the *MobileClass* is used for the mobile device service. The *PropertyClass* concentrates on the specific features which have amount and type. For instance, *CPUType*, *Speed*, *Size*, *Memory*, all kinds of the characters that illustrate the internal properties and make it easily for the clients to select.

One thing should be stated here. In spite the schema is mended by generalizing the survey on the current services, the schema is still not perfect and complete. Actually, it partly uses the reference from RDF consortium [20], simplifies and extracts the related part into the present one. Because of the critical task is to evolve the knowledge not to prefect it once, so the consummation and reasonability is not concerned during the time.

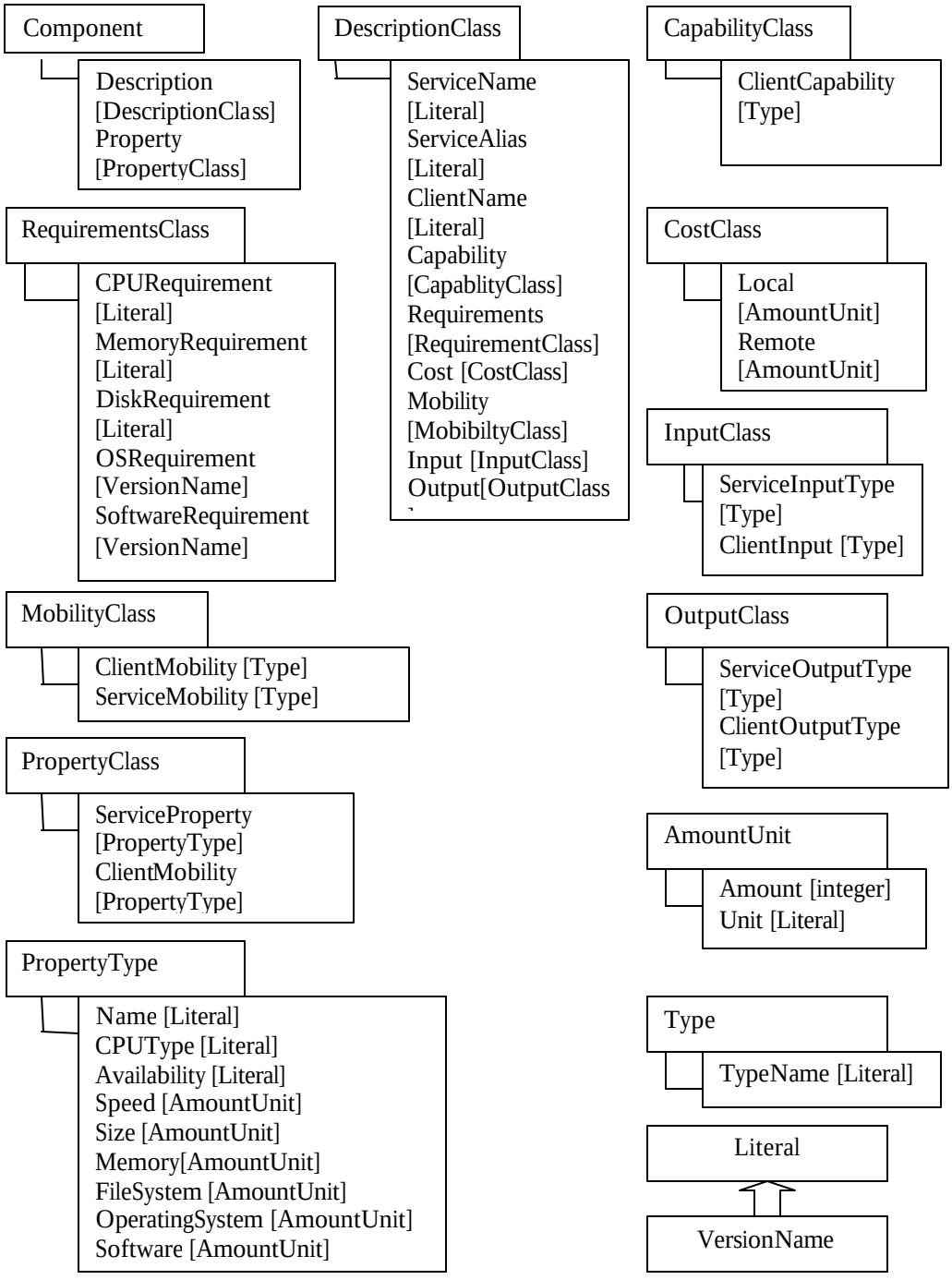


Fig.4 Generic Ontology Fundamental Schema

In figure 4, each of the square modules stands for the class and its properties resource in the RDFS hierarchy. The small square represents class with the name, and the big square below consists of the properties, while behind each property's name is the type which can be a system predefined type or neighbor class in this fundamental schema.

The schema is bewrited in RDFS language. As we have introduced it in the previous chapter, it uses classes, properties and the relations to construct the ontology. Followed in this way, every service need to use such methods to make its description, thus realizes the unification which will facilitate to dynamically change in the latter evolution.

4.4.2 Dynamic Adaptation

After the creation of fundamental schema, the evolution steps into its stage. To be dynamic means to be changing every time when there are requirements, and reflect the changes so as to have the capacity of adapting to more complex conditions. So the most important thing to be concerned in this part is what kind of requirements the system would encounter and towards each condition what kind of approaches may use to solve the conflictions and then enrich the internal knowledge base.

Classified in terms of practicality, there are six kinds of the situations would take place in the meet between the system schema and external component descriptions. These cases are generalized to cover all the possibilities that would result in temporary conflictions and require the system schema forwardly update itself and thus guarantee the unification description methods, and eventually realize the adaptation. In the following, the detail approaches are presented in turns.

An example of mobile device service, which can be thought as part of the prototype of a handheld service, is used here to demonstrate the six possible cases and corresponding solutions. Before the analysis of the situations, a "regular" description written in RDF is examined in advance. The regular here means all the classed and properties used are strictly follow the schema and has no confliction with other congeners. One thing should be stated is that these data are only for demonstrating and are not got from the manufactures, so some of them may not reasonable but they are still able to reflect the ideas.

```
<s:Component rdf:about="http://hostname/2002/compaq_pda.html">
<s:Description>
<s:DescriptionClass>
<s:ServiceName>Compaq Ipaq</s:ServiceName>
<s:ServiceAlias>3800 series</s:ServiceAlias>
<s:ClientName>EMOOSE 2002</s:ClientName>
<s:Capability>
<s:CapabilityClass>
<s:ClientCapability>1000bps</s:ClientCapability>
</s:CapabilityClass>
</s:Capability>
<s:Requirements>
<s:RequirementsClass>
<s:CPURequirement>33MHz</s:CPURequirement>
<s:MemoryRequirement>16M</s:MemoryRequirement>
<s:DiskRequirement>500M</s:DiskRequirement>
</s:RequirementsClass>
</s:Requirements>
</s:DescriptionClass>
</s:Description>
</s:Component>
```

In the following, all the cases are based on this example. And we would go over the original model from time to time to compare with the difference after re-mending the schema.

All things being described by RDF expressions are called resources, and are considered to be instances of the class *rdfs:Resource*. In the RDF schema class hierarchy, Resource is an abstract object that nearly all other objects, such as class, property and statement are its subclasses. While *rdfs:Literal* is the most primitive value type represented in RDF, typically a string of characters. The content of a literal is not interpreted by RDF itself and may contain additional XML markup. Literals are distinguished from Resource in that RDF model does not permit literals to be the subject of a statement.

Then the first possibility is related to adding a new kind of the property in the schema context, and since the type of the property to be added is critical to the final solutions, so they are divided into three different cases: add property as literal; add property of an existing class; add property of non-existing class.

4.4.2.1 Add New Property as Literal

When a new component comes, it may have its own special property although most of the descriptions are consistent with the system schema. Then the schema needs to add this new property in its corresponding class in order to fit in with the new one and enrich the descriptive ability.

In this case, we are planning to add a new property for a certain class, and the property's type is the primitive type Literal. This will not make conflictions among the former component and the new one, because the formers do not have such a property and can allow this property empty when the new schema takes into practice. For the further retrieval, since the one who omits this property item is equal to declare that it does not have such property, and the search engine would be surely pick it out from the candidates when it is executing the query match process, only examining the one who has. So in this case, the task is to add the corresponding property's name and its domain class, and simply its range class is *rdfs:Literal*.

For example, the component add a literal property under the DescriptionClass, with the name NewProperty, and its value is "Ecole des Mines", which a string in fact.

```
<s:DescriptionClass>
<s:ServiceName>Compaq Ipaq</s:ServiceName>
<s:ServiceAlias>3800 series</s:ServiceAlias>
<s:ClientName>EMOOSE 2002</s:ClientName>
<s:NewProperty>Ecole des Mines</s:NewProperty>
<s:Capability>
... ...
</s:Capability>
... ...
</s:DescriptionClass>
```

In the schema, a new property declaration is added to reflect the changes.

```
<rdf:Property rdf:ID="NewProperty">
<rdfs:domain rdf:resource="#DescriptionClass"/>
<rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>
```

The key points for the implementation: one is to find the property's domain class in the context; the other is to distinguish its literal feature apart from the other type of properties.

4.4.2.2 Add New Property of an Existing Class

In this case, the type of property which is to be added is not literal but an existing class. The existing means there is already definition for such class in the schema, and it would be easily recognized. So the task is similar to the previous one, which is adding the declaration of the new property, with the range class just next outside the tag of property.

In the example here, we create a new property in a component description, named as NewProperty, and it is used under the class DescriptionClass. The range class here refers to the existing class CapabilityClass, which is already defined in the system schema and does not need to create new one.

```
<s:DescriptionClass>
<s:ServiceName>Compaq Ipaq</s:ServiceName>
<s:ServiceAlias>3800 series</s:ServiceAlias>
<s:ClientName>EMOOSE 2002</s:ClientName>
<s:NewProperty>
<s:CapabilityClass>
<s:ClientCapability>1024bps</s:ClientCapability>
</s:CapabilityClass>
</s:NewProperty>
<s:Capability>
... ...
</s:Capability>
... ...
</s:DescriptionClass>
```

The task to update the schema is also simple. Appending a new property declaration in the schema using those already defined.

```
<rdf:Property rdf:ID="NewProperty">
<rdfs:domain rdf:resource="#DescriptionClass"/>
<rdfs:range rdf:resource="#CapabilityClass"/>
</rdf:Property>
```

The points for implementation lie on two aspects: find the property's domain class in the context; recognize the range class and examine it by comparing to the existing classes.

4.4.2.3 Add New Property of a Non-existing Class

When a new property is not a simple type but a complex class, and further, the class has not defined in the schema. Then together with the property, the corresponding class also needs to

be added. This may lead to a continual adding process because the new class has its own properties to describe the component. So it looks like a recursive process that adds new property and the new range class, and then new property rises. It would be terminated when the new property use primitive literal as its type, or an existing class as its range class. Each of these two branches ought to follow the steps stated in the previous sections.

The example here needs to add the new property, named as NewProperty2. And its range class is also a new one, which is called NewClass. Further, NewProperty3 is the property of the new class, with the type literal.

```
<s:DescriptionClass>
<s:ServiceName>Compaq Ipaq</s:ServiceName>
<s:ServiceAlias>3800 series</s:ServiceAlias>
<s:ClientName>EMOOSE 2002</s:ClientName>
<s:NewProperty2>
<s:NewClass>
<s:NewProperty3>EMN</s:NewProperty3>
</s:NewClass>
</s:NewProperty2>
<s:Capability>
... ...
</s:Capability>
... ...
</s:DescriptionClass>
```

Not only need we add the new declaration of the NewProperty2, but also add the its related class definition, and together with the consequent property.

```
<rdf:Property rdf:ID="NewProperty2">
<rdfs:domain rdf:resource="#DescriptionClass"/>
<rdfs:range rdf:resource="#NewClass"/>
</rdf:Property>

<rdfs:Class rdf:ID="NewClass">
</rdfs:Class>

<rdf:Property rdf:ID="NewProperty">
<rdfs:domain rdf:resource="#NewClass"/>
<rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>
```

Thus the tasks for implementation are to find the new property and recursively step into the tag, recognize the new class and its new properties, and repeat the previous steps until reach the end condition.

Above all the cases are related to add new property or new class if the component descriptions have difference with the system schema. Yet, sometime it would need to modify the properties, while not to add the new one. This means the component still share such property but use it in another way. For example, in the initial system schema, there is a name property to describe the name feature for every member. But a new service's name feature is more complex than ever, instead of using literal it may use other complex type of class to describe its name. It doesn't like the previous cases, because the component can not omit the original property and only to add the new one. Those new properties are towards changing the definition of the originals, but in some degree they share the same meanings and both can not be omitted. If only to add the new one and ignore the original, on the contrary, confliction would turn out. So the solution is to meliorate and make the schema suit for the new component.

Then another problem rises. If the system schema change its way, it certainly will affect the previous ones. If we give too much cares for the new components but neglect the consistence for the antecedence, the system will lose the capacity to describe them and fail to server for the further retrieval. So the solution should not only adapt to the new members, but keep the ability to consist with the previous components as well. Consequently, the scope of knowledge base will be eventually broadened.

In the following, three branch cases are taken into account, so as to demonstrate the corresponding solutions in the design of architecture.

4.4.2.4 Modify Property in Condition 1

Since in these cases the component needs to modify a certain property, it certainly will use another type of classes to redefine such property whatever they are existing or new. In this condition, the case is one component need the schema to modify a property Capability, which originally uses CapabilityClass as its range class. Simply, it does not require adding some new properties, only to change the type of the range. In order to keep consistence, our

solution here is make the new range class as the subclass of the original one. This will make the new component easily inherit the properties form the super class, and the previous components can still use the super class to describe the property without confliction with the new members, because the super class can be used at the place where requires its subclass according to the OO programming principle. Moreover, the new components may add new properties of its own in the subclass in future. Such cases would be stated in the later sections.

```
<s:DescriptionClass>
<s:ServiceName>Compaq Ipaq</s:ServiceName>
<s:ServiceAlias>3800 series</s:ServiceAlias>
<s:ClientName>EMOOSE 2002</s:ClientName>
<s:Capability>
<s:CapabilityClassSub>
<s:ClientCapability>1000bps</s:ClientCapability>
</s:CapabilityClassSub>
</s:Capability>
... ...
</s:DescriptionClass>
```

A new class with the name CapabilityClassSub is created and it inherits the property ClientCapability from the super class Capability, keep the property type as literal.

```
<rdfs:Class rdf:ID="CapabilityClassSub">
<rdfs:subClassOf rdf:resource="#CapabilityClass"/>
</rdfs:Class>
```

For the implementation, the key point is to recognize the property which has been changed into other forms especially the range, and then look up the original one and examine them to see if it is possible to derive the new class from the former.

4.4.2.5 Modify Property in Condition 2

In the previous case, the new subclass only derives from the super class, and never adds new property. Nut in most of the conditions, they change the range in order to apply some new properties of their own, and use them to describe particular features which the former schema unfeasible to express. For example, in the example above, the new component add a new property named as NewProperty under the class CapabilityClassSub which inherits from class CapabilityClass.

```
<s:DescriptionClass>
```

<s:ServiceName>Compaq Ipaq</s:ServiceName>
<s:ServiceAlias>3800 series</s:ServiceAlias>
<s:ClientName>EMOOSE 2002</s:ClientName>
<s:Capability>
<s:CapabilityClassSub>
<s:NewProperty>40Bps</s:NewProperty>
</s:CapabilityClassSub>
</s:Capability>
... ...
</s:DescriptionClass>

Then task of updating the schema is similar to the one in condition 1, but moreover it needs to add a new property declaration according to the form of the descriptions of the new component.

<rdfs:Class rdf:ID="CapabilityClassSub">
<rdfs:subClassOf rdf:resource="#CapabilityClass"/>
</rdfs:Class>
<rdf:Property rdf:ID="NewProperty">
<rdfs:domain rdf:resource="#CapabilityClassSub"/>
<rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>

In the implementation, the important thing is to extract the new property in the context and figure out its range class. Because it will involve adding property, so f the new property happens to meet the conditions mentioned before, then it should follow all the steps in the first three cases to carry out the update.

4.4.2.6 Modify Property in Condition 3

This case actually is the combination of condition 1 and 2. It needs to modify a certain property to use another new range class, and not only use some new properties in such new class but also keep some properties in the place where the original property has.

<s:DescriptionClass>
<s:ServiceName>Compaq Ipaq</s:ServiceName>
<s:ServiceAlias>3800 series</s:ServiceAlias>
<s:ClientName>EMOOSE 2002</s:ClientName>
<s:Capability>
<s:CapabilityClassSub>
<s:ClientCapability>1000bps</s:ClientCapability>
<s:NewProperty>40Bps</s:NewProperty>
</s:CapabilityClassSub>
</s:Capability>

... ...
</s:DescriptionClass>

In the example, the property Capability uses CapablityClassSub instead of the CapabilityClass and adds new property NewProperty, but it reserves the original property ClientCapability. In spite of the difference, the update solution in the schema is same as the one in condition 2.

```
<rdfs:Class rdf:ID="CapabilityClassSub">
<rdfs:subClassOf rdf:resource="#CapabilityClass"/>
</rdfs:Class>
<rdf:Property rdf:ID="NewProperty">
<rdfs:domain rdf:resource="#CapabilityClassSub"/>
<rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>
```

Similarly, the task is as well as the one in condition 2. But one more thing needs to be concern on is to distinguish the property original apart form the one to be added.

Above all the cases for possible conditions have been concerned and corresponding solutions have also been presented to conquer the problems. Actually in practice the situation would be more complex than them. But whatever complicate the case would be, they can be eventually disassembled into several small cases which are definitely tally with the ones stated before. It means the complex problem can be divided into many small ones, and by using the basic solutions, a dynamic adaptation ontology schema is successfully created and obtains the power to face many kinds of situations, and enforces fundamental of architecture to the advance retrieval task.

4.4.3 Concept Retrieval

In the architecture, the construction of ontology schema is to realize dynamic adaptation, so as the reasoning and inferencing task could take into effect. In this section, the design of retrieval and query is to be presented. This is also an important functional and operational part provided by the thesis.

Due to the query is not directly upon the RDF data but translating the all the descriptions triples into the conceptual graphs, by means of the concept retrieval, the tasks are converted to matching the sub graph in the globe graphs instead of the keyword matching. Thus the discovery can be elevated up to the semantic level, not only limited in the syntax comparison.

The query is also written by RDF, so it is hidden the intricate logical formalism to the service providers who do not know their service descriptions are not the one which would involve in the retrieval process. And the service users are also blind to this part. This will not only simplify the interaction with the users by not forcing tough mechanism, but also gain the powerful ability from the logical techniques. This is also a good solution in those domains that need data mining aids to extract the anticipant information.

The whole retrieval process can be divided into several sub processes to analyze, which can help to model and establish. Consequently, they are considered as: making the primitive into the query form; mapping the query into the Concept Graph formalism; arrange the result to the user in a human readable form. These three processes would be detail in the following sections.

4.4.3.1 Establish the RDF Query

As we present the CG formulas in the previous chapter, though it is not tough for human read, the expressions still seem to be abstract to understand. Actually, the contribution it made is to reflect the real world to the conceptual world, that is, to use the computer language to describe the objects, concepts and the relations. So is the RDF/S language. They are both about to establish the machine readable patterns to translate and extract the human concepts to the computer science domain. But they are more like the bridges linking the two worlds not totally surrender to either side. So it is not good to make the users, particularly the users who raise the query, directly face to the intricate CG formalism, and also not reasonable to force the users to establish the RDF formed data so as to embed their requests in the tags. So in this aspect, the goal is to establish the RDF query from the user request and make it better appropriate for the next mapping process.

Actually, the idea is coming from several the ontology query languages [35]. As it known to all, the necessity for building, annotating, integrating and learning tools is uncontested. However, the sole representation of knowledge and information is not enough. Human information consumers have to use and query ontologies and the resources committed to them, thus the need for ontology storage and querying tools arises. But the context of querying knowledge has changed due to the wide acceptance and use of the web as a platform for communicating knowledge. New languages for querying meta-data or data on standards, like RDF, have

emerged to enable the acquisition of knowledge from dispersed information sources, while the traditional database storage techniques have been adapted to deal with the peculiarities of the semi-structured data.

Structure Query Language (SQL) is a well known query language for the traditional relative database, and also has been widely used in the field of the data query and retrieval. While a lot of people introduce SQL into the domain of ontology description, and try to adapt and meliorate it to do help to the query on the semantic data. For example, the SquishQL [22], developed by ILRT Semantic Web research Group [23], is an SQL-style, statement-based query language that supports the RDF model and syntax. Being a simple graph-navigation query language for RDF based on a sub graph matching mechanism, SquishQL uses SQL-like constructs to reflect RDF's graph syntax. Apart from supporting a query model based on a graph pattern formed from variables for nodes, arc and literals, it introduces filter functions in the form of Boolean expressions over the variables. Thus, in SquishQL there are two classes of constrains: patterns and filter expressions. The pattern language is formed from triple patterns <subject, predicate, object> describing edges of the graph and a variable or an explicit value. For each component of a triple, i.e., subject, predicate and object it allows either a variable or an explicit value. Filter functions restrict the values that the variables over the components of a triple can take. In general, the patterns are generative, since they created bindings and the filters are restrictive, in view of the fact an SquishQL query by specifying the graph pattern as a list of triple patterns and the AND clause corresponds to the restrictive part, which specifies the Boolean expressions.

SquishQL supports a considerable functionality for RDF query expression and has formed the basis of a number of RDF query languages of diverse complexity. A language derived form SquishQL is RDQL, which is being developed by HP Semantic Web Group [24]. RDQL is a syntax and query API whose purpose is to act as a model-level access mechanism. It extracts information from an RDF model by treating RDF as data and providing query triple patterns and constrains over a single RDF model.

Let us take a brief look at the syntax and grammar of the RDQL, and figure out the advantages that we can take such query language of and contribute to our solution.

```
<rdf:RDF xml:lang="en"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:s="http://localhost/2002/services.rdfs#">
<s:Component rdf:about="http://localhost/2002/compaq.html">
<s:Description>
<s:DescriptionClass>
... ...
<s:Requirements>
<s:RequirementsClass>
<s:CPURequirement>33MHz</s:CPURequirement>
<s:MemoryRequirement>16M</s:MemoryRequirement>
<s:DiskRequirement>500M</s:DiskRequirement>
</s:RequirementsClass>
</s:Requirements>
... ...
</s:DescriptionClass>
</s:Description>
</s:Component>
</rdf:RDF>
```

Here is an example of the requirement descriptions of a service component, which provide some basic hardware information. Now if we construct a RDQL query to find the resource whose DiskRequiremnet is 500M, the query should be written in this way.

```
SELECT ?resource
WHERE (?resource, <s:DiskRequirement>, ?amount)
AND ?amount = 500M
USING s FOR <http://localhost/2002/services.rdfs#>
```

It is obviously using such a convenient query language, the process of making the request would be pretty simple and even the query sentences are close to the common human language, which is more readable and comprehensive. So the idea to analyze the RDF query language here is to take use of some talent features and facilitate the query for the user, so as to prevent the user directly face to the tough ontology structure.

Another thing should be concerned. Never do we use such kind of the query language directly retrieval inside the RDF data repository, this is due to we use CG reasoning and inferencing, not the simple triple matching as those query languages do. The key point we use some SQL-like query syntax is to make a smooth tie between the user and the process of mapping RDF to CG, which will avoid the user or the programmer directly dealing with complex work. Moreover, this feature would even hide the knowledge of those techniques as

we propose in the future work. Generally, there are two main reasons for introducing the idea into the query work:

- ✧ It can offer a human readable and comprehensive interface for the users to construct their queries.

- ✧ It will be easy to use the query to establish the RDF structured query tags, because the query actually consists of a lot of information related to the ontology. And it is also due to the mapping process needs it to be the source data.

To not separate the pertinence in the query construction period, how the reference the query language is used would be present together with the mapping work in the next section.

4.4.3.2 Mapping Query to CG

The main interest of mapping RDF to CG is the adequacy between the two models, i.e. concepts and relations smoothly map onto classes and properties that are defined independency in CG as well as in RDF. Furthermore, it enables us to use RDF without any knowledge of Conceptual Graphs.

The second reason is the relevance of the CG projection operation to querying a RDF/CG base. Querying RDF metadata consists of the retrieving RDF triples belonging to classes, taking specialization into account. This can be done through the projection operation. Moreover, thanks to the implementation platform that we have chosen, namely O. Corby, Corese [25], it is possible to parameterize precisely the graph matching process. Hence, it is possible to tune concept matching, including type and instance matching. Relation matching can also be parameterized, as well as other aspects of the graph matching. This functionality is well adapted to meta-data information retrieval as it authorizes approximate matching along specialization and generalization on relations and concepts.

As stated before, the query language is RDF itself. The query should be a partial RDF statement that the user is looking for. The query may hold variables, prefixed by "?", to indicate the parts that are known, the value of which should be returned by the query processor. Let us construct the query RDF statement according to the example in the last

section. We want to know the service whose name is "Compaq", and then the SQL-like sentence is:

```
SELECT ?resource
WHERE (?resource, <s:Component>, ?b),
(?b, <s:Description>, ?c),
(?c, <s:DescriptionClass>, ?d),
(?d, <s:ServiceName>, 'Compaq')
USING s FOR < http://localhost/2002/services.rdfs#>
```

the RDF statement is like:

```
<s:Component rdf:about ='?resource' s:Description='?d'/>
<s:DescriptionClass rdf:about='?d' s:ServiceName='Compaq'/>
```

Then the RDF query is translated into the graph shown below:

```
[Component] –{
-> (Description) -> {[DescriptionClass] –{
-> (ServiceName) -> [Literal : Compaq]}
```

The query processor projects the query graph on the CG base. The resulting (sub) graphs are translated back into RDF in order to be presented to the user in a uniform way.

More conveniently, the Corese API also implement approximate search on literal values, so an approximate comparator that tests whether the query literal value is included into the graph literal value. Approximate query values are prefixed by the "~" character.

It is then possible to send a query that searches the ClientName, the value of which contains "2002", as shown below.

```
<s:Component rdf:about ='?resource' s:Description='?d'/>
<s:DescriptionClass rdf:about='?d' s:ClientName='~2002'/>
```

It is obvious the primitive query is suitable for the person to make because the SQL-like sentence is close to the human language. In other hand, the RDF query established according to the user query is pretty easy to be translated into CG formalism. So by combining two processes, it really facilitates the concept retrieval.

4.4.3.3 Return Result

As the resulting graphs are finally translated back into RDF, they would be present in the uniform way as its initials. It is certainly, the user would not accept a RDF statement as the result, which is not suitable. So an addition extraction process is executed so as to present the user only the available information, getting rid of redundancy. For example the when it returns the service name, the redundant RDF syntax and XML tags would be trimmed off, and only the literal name is given as a string and transmitted to the user. Other cases are same to this.

Above is the design of concept retrieval, and the processes of the concept retrieval are illustrated and made examples of. Together with dynamic adaptation, they are the core of the architecture and central content of the design in the thesis.

4.4.4 Smart Device Service Discovery

To have an integrated scenario to take the system into effect, the application at the client side is needed to design and implement. We choose a PDA scenario which the smart device looks up for the available printer service or discovers other available congeners to exchange information in the local network. The infrastructure would be established on the based of .NET Compact Framework, and using the Smart Device Extensions to be the developing environment. The specifications of such two techniques and the advantages they provide have already been presented in the Chapter 3.2.7. Here we will show how to make use of them in our scenario and integrate into our architecture.

As we known, one great talent feature in .NET is different kinds of programming languages can run together and interact with each other on the same infrastructure as long as the platform is supported by the .NET framework. Moreover, in SDE, it can use some computer programming languages like C#, VB to make the handheld application for PDA. This will greatly contribute to the development, and realize the unification across the platforms.

Our idea for the design in this part lies in two aspects: one is to allow the client send query request to the server; the other is to deliver the query to the executive part in the system. The first aspect is easy to think of, for there must be a portal for the distributed device to obtain the information from others. While the second one is due to the limitation of the architecture,

because in the current design, the executive parts, such as dynamic adaptation and concept retrieval, are implemented by Java. It is certainly not compatible to the .NET infrastructure. The detail reason for this inconsistence limitation would be explained in the next chapter. But here this linking problem has to be considered during the design.

Web Service could be a good approach to serve as the portal to collect the requests, and it is easy to combine with the other application, thus improving the extensibility in the network. Moreover, Web Service is the most promoted technology in .NET and solid supported. As the perspective in this thesis is to turn the whole prototype into the .NET in the future work, introducing the Web Service into the architecture is reasonable and effective.

To the second problem stated previously, actually the essence of problem is communication between the portal web service and service discovery system. Currently, the solution is temporary through socket transmission. When the request query is coming, the portal service delivers it to the service discovery system. And after the result comes out, the portal service sends it back to the client device as the final response.

4.4.5 Overall Architecture

After describing each part of the architecture, it is time to give a global view of the whole work. By presenting the overall architecture, we will see how the constituent elements have been designed to support dynamic service discovery.

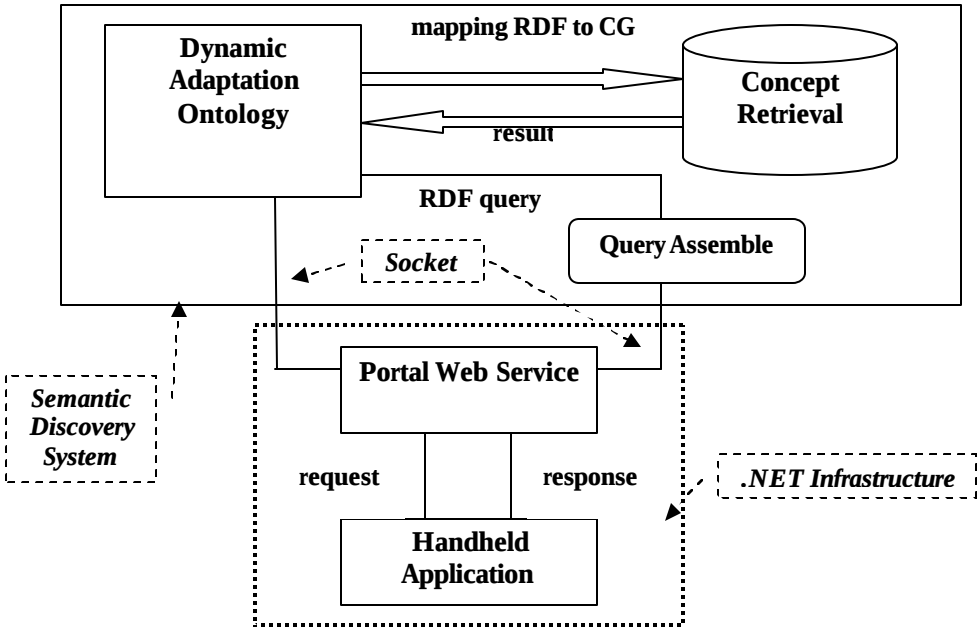


Fig. 5 Overall Architecture

Figure 5 shows the overall structure of the architecture. The whole work can approximately divided into two parts: client service part and discovery server part. The former one is constructed in .NET infrastructure making use of web service as portal on server and Compact Framework as development environment on client. The latter part consists of two portions, one is dynamic adaptation that constitutes the system ontology, and the other is concept retrieval that support CG based reasoning and inferencing. Between the web service portal and the discovery system, there is a query assemble function modal, which is to translate the user SQL-like query into RDF query so as facilitating the continual discovery procedure. In addition, the two parts currently are using socket to communicate with each other, and this would be improved or redesigned in future work due to its insufficiency.

Chapter 5 Implementation

This chapter is to present the detail implementation of architecture described above. Since the architecture could be divided into several portions and each of them have its own operation and functionality which is relatively independent. In the following we will first state the reasons which result in the partly changes on selection of the programming platform. Then we demonstrate how each of the modules implemented and selectively choose the core classes of functional components in order to give a clear overview on the prototype. Company with that, a terse walkthrough scenario will also be presented so as to emphasis the feasibility acquired by the aborative design.

5.1 Programming Platform

As it shows in the figure 5, the discovery system part which is comprised of two modules, ontology construction and concept retrieval, is actually implemented upon the Java infrastructure currently. Although the initial purpose is to perspective the whole prototype constructed in the .NET platform as much as possible, it is thought to adequately make use of the ample programming resource, and take advantage of the talent features from the framework. But at last the point of goal is not thoroughly carried through. The fact lies in three main reasons.

The first reason is because .NET is a very new born programming platform, in some research domains of computer science it still has empties and rarely deals with the practical problems. While these domains have been made research on for a long time and it exists some mature solutions to carry out and help to the future advanced researches to base on. So we can say it is too new to handle every aspects of the computer science research. There is less support could be found to contribute the thesis, and those techniques which meet the requirements and can integrated into the architecture are almost implemented in other platforms.

For example, in the module of conceptual retrieval, Corese is a good foundation to realize the mapping task between RDF and CG, which provides a Java API to program and facilitates the efficiency. But as it known to all, Java platform is isolated inside its virtual machine and not easy to be combined with other platform, especially .NET. Moreover, even if we go

beyond this part and use its idea to make our own mapping module, we still need to face to the CG domain, which is totally blank to the .NET programming. And the matter of the fact is, Corese itself is implemented on as CG infrastructure, named as NOTIO [26] [27], which is also a Java based work on CG expression and construction.

Another example is since we use RDF/S as the ontology description language, it certainly needs a language parser to check the validation of syntax and grammar. This is a common but necessary step in the language analysis. Unfortunately, currently most of the effective RDF/S parsers are developed in Java, this is due to open source policy is popular in Java world. But .NET is definitely weak on this subject, presently there is no related parsers can be used in .NET. So it is no choice but to turn to Java platform to carry out the prototype in the part of work. If we leave off all the ready-made techniques and build all the work by self, it will go beyond the title of this thesis, and improper to the six-month research term.

The second reason is for the architecture itself. Because the design of the prototype is in a higher level than implementation, the main idea is not wholly platform interrelated. So we believe as the future work continuing on, the entire architecture could be transformed into .NET platform. And it would also be the future work discussed in the next chapter. So it is reasonable to complete some part in Java as a temporary solution.

The last reason can be see as the complement to not choosing .NET in the system part development. Because we adopt .NET Compact Framework to develop the client side application and it takes benefit from the platform, it completes the whole procedure, and thus a walkthrough scenario could be executed to be demonstrated later in this chapter.

In a word, the selection of the platform is effected by several external factors, but it will not impair the thought of design, and the implementation is still able to illustrate the original concepts.

5.2 Module Specification

In this section, the core modules of the architecture are selected to be specified. They will be list according to the module part it belongs to in the system structure.

5.2.1 Semantic Discovery System

This part is mainly developed for the major functionality. By using the API form VRP infrastructure [28], which is an effective RDF parser developed by ICS-FORTH [29], we can carry out both lexical and syntax check and create an internal ontology model for advanced analyze. Further, to figure out the new emerging component's description, we have created the corresponding classes to validate the semantics and reflect the difference to the update function to mend the schema, thus realizing dynamic adaptation.

Above all, the basic schema is constructed and ready for reference. When a new service component is introduced into the system, we first check its RDF description on lexical and syntax validation. If there are some errors, the further procedures will not be executed and the errors would be thrown out. If the description passes the check, a semantic validating will be carried on to see if it follows with the system current ontology schema. Having distinguished the discrepancy among the new component and ontology schema, it is time for the schema to deicide if it is necessary to update. And the update cases would be complex, but any of them can be recognized in term of the conditions presented in chapter 4, although maybe over two of the case solutions need to be used together. After the schema has been successfully reviewed, the new component can be analyzed again, that is to say, the RDF data can be re-parsed to guarantee now the service is completed accepted by the system, so that the ontology has succeed to enrich its knowledge base.

In the other part of the discovery module, the concept retrieval core class firstly loads the RDF schema into the internal model, and together the RDF data which to be retrieved are also loaded to project into CG graph. After finishing loading schema and data, the class is ready for query. Then when the RDF query comes, it is also projected as a sub graph according to CG formalism, and it would be compared in the global graph which is similar to matching the keyword in the database. After reasoning and referencing in terms of the schema semantics, the result would be returned to the adaptation module, and finally the arranged result would be sent back to the client.

In addition, the initial query comes from the client side is primitively SQL-like codes, there is a pre-procedure functional component on the way of it to the discovery system. It extracts the useful information form the query and converts them into RDF style, so that to facilitate

the later mapping process. This function module is automatically engaged, without system intervening.

5.2.2 Portal Web Service

We create a web service using C# to server as the portal for the handheld device querying under .NET. The main function is to accept the external query form the distributed ubiquitous devices and deliver it to the semantic service discovery system. And vice versa it sends the query result as response to the client, thus finishes a complete round of query process. To make it as a web service is not only because of the benefited convenience, but it would be easy to be utilized by the application on the smart device in the network environment. Further it is also consistent of the goal to make use of .NET adequately.

The communication methods between the portal service and the discovery system is through socket message, while it is a temporary solution for the current architecture and would be re-considered and find a good approach to solve the problem of information exchange between different heterogeneous platforms.

5.2.3 Handheld Client

The final implementation of client side application is made on a PDA simulator provided by Smart Device Extension in Visual Studio .NET [30]. The simulator is running under Pocket PC [31] operating system which is part of the Windows CE [32] platform for PDA device. This will truly demonstrate the practical performance in the walkthrough scenario. And it combines the portal web service and offers a terse interface to the user to create the query and examine the result. Since the goal is not to making a brilliant interface, but to simulate the handheld application scenario, so we do not emphasize on this and just to offer the functions.

5.3 A Walkthrough Scenario

The following description is used to walk through a typical scenario for demonstrate a complete procedure for service adaptation and service discovery. This will make the system's functionality and operational characteristics in more detail. It will give the reader a clear overview on the mechanisms which are designed to achieve the goals by the architecture, and thus prove the feasibility of this service discovery system.

Since there are two roles would interact with the system, and normally each of their active procedures are separated, so here the example actually is comprised of two sub scenarios: one is the service providers introduces their own service component to the dynamic adaptation module, and the system re-builds the ontology schema to adapt the discrepancy; the other is handheld user makes query on the client side, and the system handles the request and executes the searching mechanism to perform the semantic discovery on the cognizant services.

5.3.1 Service Provider Oriented Scenario

Suppose there is a service provider who can offer a print service on the network, and the service can be described as a HP postscript printer component. And we assume the access point for this service is through a HTTP link <http://hostname/2002/printer/hp>, which means when the retrieval mechanism fix on this print service as the target service, then the user will receive this link to access it remotely. And some further information exchange could continue and established directly between the service and user not through the discovery system. This is not cared in this thesis. Here is the a brief RDF description of the print service.

```
<rdf:RDF xml:lang="en"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:s="http://hostname/2002/services.rdfs#">

<s:Component rdf:about="http://hostname/2002/printer/hp">
<s:Description>
<s:DescriptionClass>
<s:ServiceName>HP Printer</s:ServiceName>
<s:ServiceAlias>IS6000</s:ServiceAlias>
<s:ClientName>EMOOSE 2002</s:ClientName>
<s:Capability>
<s:CapabilityClass>
<s:ServiceCapability>
<s:Type>
<s:TypeName>Postscript</s:TypeName>
</s:Type>
</s:ServiceCapability>
</s:CapabilityClass>
</s:Capability>
<s:Requirements>
<s:RequirementsClass>
```

<s:CPURequirement>233MHz</s:CPURequirement>
<s:MemoryRequirement>32M</s:MemoryRequirement>
<s:DiskRequirement>1G</s:DiskRequirement>
</s:RequirementsClass>
</s:Requirements>
</s:DescriptionClass>
</s:Description>
</s:Component>

</rdf:RDF>

Comparing with the fundamental schema in figure 4, we can find in the description of this new component, under the class CapabilityClass there is a new property named as ServiceCapability which does not exist in the system schema. So as to make a consistent ontology to generalize the concepts among the different services components, it will forwardly add this new property and its related information and refresh the check process to validate the changes. The new item to be added would look like as follows:

<rdf:Property rdf:ID="ServiceCapability">
<rdfs:domain rdf:resource="#CapabilityClass"/>
<rdfs:range rdf:resource="#Type"/>
</rdf:Property>

Then the new schema would be ready to be inferenced for all the components no matter they are using the new property whether or not.

5.3.2 Handheld User Oriented Scenario

Assume there is a handheld device user, who wants to print a document stored in his PDA. Since his document is postscript formatted, so the query he will make is to search available print service that is compatible with postscript. The primitive query is made SQL-like which is comprehensive for the human user to understand.

SELECT ?resource ?n
WHERE (?resource, <s:Component>, ?a), (?a, <s:Description>, ?b),
(?b, <s:DescriptionClass>, ?c),(?c, <s:ServiceName>, ?n), (?c, <s:Capability>, ?d),
(?d, <s:CapabilityClass>, ?e), (?e, <s:ServiceCapability>, ?f), (?f, <s:Type>, ?g),
(?g, <s:TypeName>, 'Postscript')
USING s FOR <http://hostname/2002/services.rdfs#>

We can see there are several internal variables used to link tags in the hierarchy, which are not cared in the view of user. Only the variable for the ServiceName and resource of Component

are needed to return as result. This query is firstly sent to the portal web service, and the latter delivers it into the semantic discovery system. During the transmission, it is required to be converted into the RDF style.

```
<s:Component rdf:about ='?resource' s:Description='?b'/>
<s:DescriptionClass rdf:about='?b' s:ServiceName='?n' s:Capablity='?c'/>
<s:CapablityClass rdf:about='?c' s:ServiceCapability='?d'/>
<s:Type rdf:about='?d' s:TypeName='Postscript'/>
```

And this RDF query will be sent to the ontology research part, mapped into CG formalism and then the retrieval mechanism executes searching and inferencing process. At last, the result is returned and rearranged in a readable style for the user.

{http://hostname/2002/printer/hp, HP Printer}

The client can establish direct communication with the result printer in terms of the resource link, and further message exchange and the final service consumption would not need the involvement of the discovery system, so the search process is complete.

5.4 Limitations & Future Work

With the approach taken to evaluate the architecture we believe that the model is still in an early stage, there are still limitations towards improving.

A major limitation has already been mentioned before, that is the overall architecture has not entirely constituted upon the .NET platform. Though we have explain the main reasons for this limitations in the chapter 5, the continuous work should be still carry on, and we also believe with the development of .NET there would be more and more relative work and techniques given birth to, and the powerful and talent advantages would eventually emerge so long as we remodel the prototype and reflect the architecture into the well-rounded programming platform.

The other one is the information exchange among each of the functional modules. Currently, the variance still exists in seaming the different parts into an integrated architecture, especially in linking up the portal web service and the core discovery system. Partly, it is due to our

using two different platforms. But by conquering the previous limitation, this deficiency should have a good solution to eliminate.

Another limitation is that although the system fundamental schema is established according to a precise survey on several exemplifications of different RDF research groups and also stipulated from the W3C consortium criterion, the descriptive ability of current ontology is still weak in some extend. This is not only due to the diversity of the increasing services, but also the specifications of the ontology description languages are still in development in order to consummate themselves to achieve a high level performance. So in the future, we would keep eyes on the latest progress in the ontology research field, and try to make our work more stable, robust, and efficient.

Chapter 6 Conclusion

The current trends on ubiquitous computing have created new requirements for discovering and using the available services in the network. This leads to require semantic interoperability between heterogeneous entities, which means to realize dynamic ontology to exchange semantic information and configure automatically.

This thesis proposes a generic approach to the semantic service discovery that allows creating dynamic adaptive ontology to facilitate the description of service components, and using conceptual retrieval as a powerful reasoning engine to acquire high performance on searching process. A handheld application is also designed and implemented to simulate the practical query scenario. The overall architecture uses several existing techniques such as ontology description language RDF/S, conceptual retrieval engine CG, RDF mapping tools Corese, .NET Web Service technology and Smart Device Extension environment in .NET platform. Making use of these advanced techniques, the architecture is designed on a solid foundation, and the high level module-independent. This would also benefit to extend current prototype in the future work.

Another major contribution of this thesis is a comprehensive survey of all major research being conducted in the area of semantic service discovery. The applicability of many of the techniques to ontology construction is also discussed.

By means of the rapid development for mobile device in .NET, the final implantation manages to handle the general query scenario, and thus realized the intelligent search in the ubiquitous computing area in a certain extent. Generally, the idea in this thesis will impact in the future way of building service discovery for increasing amount of the services in the Internet.

BIBLIOGRAPHY

[1] E. Guttman, C.P. 1999. RFC 2608: Service Location Protocol v.2 Draft. Technical report, Sun Microsystems

[2] J. Rekesh. 1999. UPnP, Jini and Salutation – A look at some popular coordination framework for future network devices. Technical report, California Software Labs

[3] The Salutation Consortium Inc. 1999. Salutation Architecture Specification (Part-1), version 2.1 edition

[4] Chen H. 2000. *Developing a Dynamic Distributed Intelligent Agent Framework Based on the Jini Architecture*, University of Maryland Baltimore

[5] Microsoft Corporation. 1999. *Universal Plug and Play device Architecture Reference Specification*, version 0.9 edition

[6] D. Fensel *Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce*, Springer-Verlag, 2000

[7] I. Horrocks, D. Fensel, J. Broekstram, S, Decker, M. Erdmann, C. Goble, F. Van Harmelen, M.Klein, S.Staab, and R. *Studer OIL: The Ontology Interence Layer*, 2000

[8] A. Newell *The Knowledge Level, Artificial Intelligence*, 18:87-127. 1982

[9] D. Fensel, I. Horrocks, F. Van Harmelen, S. Decker, M. Erdmann, and M. Klein OIL in a Nutshell. *Proceedings of the Workshop on Applications of Ontologies and problem-solving Methods*, 14thg European Conference on Artificial Intelligence ECAI'00, Berlin, Germany August 20-25, 2000

[10] R. Mizoguchi, M. Ikeda. *Towards Ontology Engineering*, Technical Report AI-TR_96-1, The Institute of Scientific and Industrial Research, Osaka University, 1996.

[11] T. R. Gruber. *What is an Ontology?* <http://www-ksl.standford.edu/kst/what-is-an-onotlogy.html>

[12] O. Lassila, R. Swick. *Resource Description Framework (RDF) Model and Syntax Specification*, W3C Recommendation. February 1999.

[13] D. Brickley, R.V. Guha. Resource Description Framework (RDF/S) Specification 1.0, W3C Candidate Recommendation. March 2001.

[14] Extensible Markup Language (XML), W3C Architecture Domain <http://ww.w3.org/XML>

[15] T. Berners-Lee, R. Fielding, L. Masinter. Uniform Resource Identifiers (URI): Generic Syntax, RFC 2396, August 1998. <http://www.ietf.org/rfc/rfc2396.txt>

[16] T. Bray, D. Hollander, A. Layman. *Namespaces in XML*, W3C Recommendation. January 1999.

[17] P. Martin, P. Eklund. *Embedding Knowledge in Web Documents: CGs versus XML Metadata Languages*. Proc. Of the 7th Int. conf. on conceptual Graphs (ICCS'99), SpringerVerlag, August 1999.

[18] P/ martin and P.Eklund. *Embedding Knowledge in Web Documents*. Proc. Of the 8th Int. World Wide Web Conf. (WWW8), p. 324-341, Elsevier, 1999.

[19] D. Brickley and R.V Guha. Resource Decription Framework (RDF) Schema specification. W3C proposed recommendation, March 1999. <http://www/w3.org/TR/1999/PR-rdf-schema-1990303>

[20] O. Corby, R. Dieng, and C. Henert AConcetual *Graph Model for W3C Resource Description Framework*, International Conference on Conceptual Structures, ICCS2000, Darmstadt, August 2000.

[21] Resource Description Framework (RDF) Consortium, <http://www.w3.org/rdf>
[22] L. Miller, A. Seaborne, A. Reggiori. *Three Implementations of SquishQL, a Simple RDF Query Language*, 1st International Semantic Web Conference (ISWC2002), June 2002. Sardinia, Italy.
[23] ILRT: Institute for Learning and Research Technology, University of Bristol <http://ilrt.org/discovery/2001/02/squish/>
[24] RDF Data Query Language, HPL Semantic Web activity <http://www.hpl.hp.com/semweb/rdql.html/>
[25] O. Corby, *Corese: A COnceptual REsource Search Engine* < http://www-sop.inria.fr/acacia/soft/corese.html>
[26] M. Chein, M. L. Mugnier. *Conceptual Graphs : Fundamental Notion*. Revue d Intelligenc Artificlle, vol. 6, n. 4, 1992
[27] J. F. Sowa. *Conceptual Structures: Information Processing in Mind and Machine*. Addison-Wesley, Reading, 1984
[28] K. Tolle, *Validating RDF Parser: A Tool for Parsing and Validating RDF Metadata and Schemas*, Master's Thesis, Univ. of Hannover
[29] ICS-FORTH RDFSuite, <http://athena.ics.forth.gr:9090/RDF/VRP/>
[30] L. Roof *Develop Handheld Apps for the .NET Compact Framework with Visual Studio .NET*, MSDN magazine, October 10, 2001
[31] Pocket PC, Mobile Device <http://www.microsoft.com/mobile/>
[32] Windows CE.NET, The realtime operating system for rapidly developing smart mobile devices <http://www.microsoft.com/windows/embedded/ce.net/>
[33] P. Hayes, *RDF Model Theory*, W3C Working Draft, April 2002 <http://www.w3.org/TR/2002/WD-rdf-mt-20020429/>
[34] J. Heflin, R. Volz, J. Dale *Requirements for a Web Ontology Language*, W3C Working Draft March 2002 <http://www.w3.irg/TR/2002/WD-webont-req-20020307/>
[35] A. Magkanaraki, G. karvounarakis, T. Anh, V. Christophides, D. Plexousakis, *Ontology Storage and Querying*, Technical Report, April 2002
[36] C. Bettstetter, C. Renner, *A Comparison of Service Discovery Protocols and Implementation of the Service Location Protocol*, TUM Institute of Communication networks, Munich, Germany
[37] H. Chen, D. Chakraborty, L. Xu, A. Joshi, T. Finin, *Service Discovery in the Future Electronic Market*, Department of Computer Science, University of Maryland Baltimore County
[38] D. Trastour, C. Bartolini, J. Gonzalez-Castillo, *A Semantic Web Approach to Service Description for Matchmaking of Services*, HP Labs
[39] D. Chakraborty, F. Perich, S. Avancha, A. Joshi, *DReggie: Semantic Service Discovery for M-Commerce Applications*, Department of Computer Science, University of Maryland Baltimore County
[40] R. McGrath, M. D. Mickunas, R. H. Campbell, *Semantic Discovery for Ubiquitous Computing*, National Center for supercomputing Applications, university of Illinois, Urbana-Champaign
[41] D. Touzet, J.-M. Menaud, Fr. Weis, P. Couderc, M. Banâtre. *SIDE Surfer: a Spontaneous Information Discovery and Exchange System*, 2nd international Workshop on Ubiquitous Computing and Communication, Barcelona, Spain, September 2001.

Appendix A: RDF Schema for Service Description

This is the entire RDF schema composed to describe service components and establish the fundamental ontology of the system in this thesis.

```
<?xml version="1.0" ?>
<rdf:RDF
    xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs  = "http://www.w3.org/2000/01/rdf-schema#"
    xmlns:xs = "http://www.w3.org/1999/XMLSchema-datatypes#"
>

<rdfs:Class rdf:ID="Component">
    <rdfs:label>Component</rdfs:label>
    <rdfs:comment></rdfs:comment>
</rdfs:Class>

<rdf:Property rdf:ID="Description">
    <rdfs:domain rdf:resource="#Component"/>
    <rdfs:range rdf:resource="#DescriptionClass"/>
</rdf:Property>

<rdfs:Class rdf:ID="DescriptionClass"/>

<rdf:Property rdf:ID="ServiceName">
    <rdfs:domain rdf:resource="#DescriptionClass"/>
    <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>

<rdf:Property rdf:ID="ServiceAlias">
    <rdfs:domain rdf:resource="#DescriptionClass" />
    <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>

<rdf:Property rdf:ID="ClientName">
    <rdfs:domain rdf:resource="#DescriptionClass"/>
    <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
</rdf:Property>

<rdf:Property rdf:ID="Capability">
    <rdfs:domain rdf:resource="#DescriptionClass"/>
    <rdfs:range rdf:resource="#CapabilityClass"/>
</rdf:Property>

<rdf:Property rdf:ID="Requirements">
```

```
    <rdfs:domain rdf:resource="#DescriptionClass"/>
    <rdfs:range rdf:resource="#RequirementsClass"/>
</rdf:Property>

<rdf:Property rdf:ID="Cost">
    <rdfs:domain rdf:resource="#DescriptionClass"/>
    <rdfs:range rdf:resource="#CostClass"/>
</rdf:Property>

<rdf:Property rdf:ID="Mobility">
    <rdfs:domain rdf:resource="#DescriptionClass"/>
    <rdfs:range rdf:resource="#MobilityClass"/>
</rdf:Property>

<rdf:Property rdf:ID="Input">
    <rdfs:domain rdf:resource="#DescriptionClass"/>
    <rdfs:range rdf:resource="#InputClass"/>
</rdf:Property>

<rdf:Property rdf:ID="Output">
    <rdfs:domain rdf:resource="#DescriptionClass"/>
    <rdfs:range rdf:resource="#OutputClass"/>
</rdf:Property>

<rdfs:Class rdf:ID="CapabilityClass"/>

<rdf:Property rdf:ID="ClientCapability">
    <rdfs:domain rdf:resource="#CapabilityClass"/>
    <rdfs:range rdf:resource="#Type"/>
</rdf:Property>

<rdf:Property rdf:ID="ServiceCapability">
    <rdfs:domain rdf:resource="#CapabilityClass"/>
    <rdfs:range rdf:resource="#Type"/>
</rdf:Property>

<rdfs:Class rdf:ID="RequirementsClass" />

<rdf:Property rdf:ID="CPURequirement">
    <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
    <rdfs:domain rdf:resource="#RequirementsClass" />
</rdf:Property>

<rdf:Property rdf:ID="MemoryRequirement">
    <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
    <rdfs:domain rdf:resource="#RequirementsClass"/>
</rdf:Property>
```

```
<rdf:Property rdf:ID="DiskRequirement">
    <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
    <rdfs:domain rdf:resource="#RequirementsClass"/>
</rdf:Property>

<rdf:Property rdf:ID="OSRequirement">
    <rdfs:domain rdf:resource="#RequirementsClass"/>
    <rdfs:range  rdf:resource="#VersionName"/>
</rdf:Property>

<rdf:Property rdf:ID="SoftwareRequirement">
    <rdfs:domain rdf:resource="#RequirementsClass"/>
    <rdfs:range  rdf:resource="#VersionName"/>
</rdf:Property>

<rdfs:Class rdf:ID="CostClass"/>

<rdf:Property rdf:ID="Local">
    <rdfs:domain rdf:resource="#CostClass"/>
    <rdfs:range rdf:resource="#AmountUnit"/>
</rdf:Property>

<rdf:Property rdf:ID="Remote">
    <rdfs:domain rdf:resource="#CostClass"/>
    <rdfs:range rdf:resource="#AmountUnit" />
</rdf:Property>

<rdfs:Class rdf:ID="MobilityClass" />

<rdf:Property rdf:ID="ClientMobility">
    <rdfs:domain rdf:resource="#MobilityClass" />
    <rdfs:range rdf:resource="#Type" />
</rdf:Property>

<rdf:Property rdf:ID="ServiceMobility">
    <rdfs:domain rdf:resource="#MobilityClass" />
    <rdfs:range rdf:resource="#Type" />
</rdf:Property>

<rdfs:Class rdf:ID="InputClass" />

<rdf:Property rdf:ID="ServiceInputType">
    <rdfs:domain rdf:resource="#InputClass" />
    <rdfs:range rdf:resource="#Type" />
</rdf:Property>

<rdf:Property rdf:ID="ClientInputType">
    <rdfs:domain rdf:resource="#InputClass" />
```

```
    <rdfs:range rdf:resource="#Type" />
</rdf:Property>

<rdfs:Class rdf:ID="OutputClass" />

<rdf:Property rdf:ID="ServiceOutputType">
    <rdfs:domain rdf:resource="#OutputClass" />
    <rdfs:range rdf:resource="#Type" />
</rdf:Property>

<rdf:Property rdf:ID="ClientOutputType">
    <rdfs:domain rdf:resource="#OutputClass" />
    <rdfs:range rdf:resource="#Type" />
</rdf:Property>

<rdf:Property rdf:ID="Property">
    <rdfs:domain rdf:resource="#Component" />
    <rdfs:range rdf:resource="#PropertyClass" />
</rdf:Property>

<rdfs:Class rdf:ID="PropertyClass" />

<rdf:Property rdf:ID="ServiceProperty">
    <rdfs:domain rdf:resource="#PropertyClass" />
    <rdfs:range rdf:resource="#PropertyType" />
</rdf:Property>

<rdf:Property rdf:ID="ClientProperty">
    <rdfs:domain rdf:resource="#PropertyClass" />
    <rdfs:range rdf:resource="#PropertyType" />
</rdf:Property>

<rdfs:Class rdf:ID="PropertyType" />

<rdf:Property rdf:ID="Name">
    <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal" />
</rdf:Property>

<rdf:Property rdf:ID="CpuType">
    <rdfs:range rdf:resource="#Type" />
</rdf:Property>

<rdf:Property rdf:ID="Availability">
    <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal" />
</rdf:Property>

<rdf:Property rdf:ID="Speed">
    <rdfs:range rdf:resource="#AmountUnit" />
```

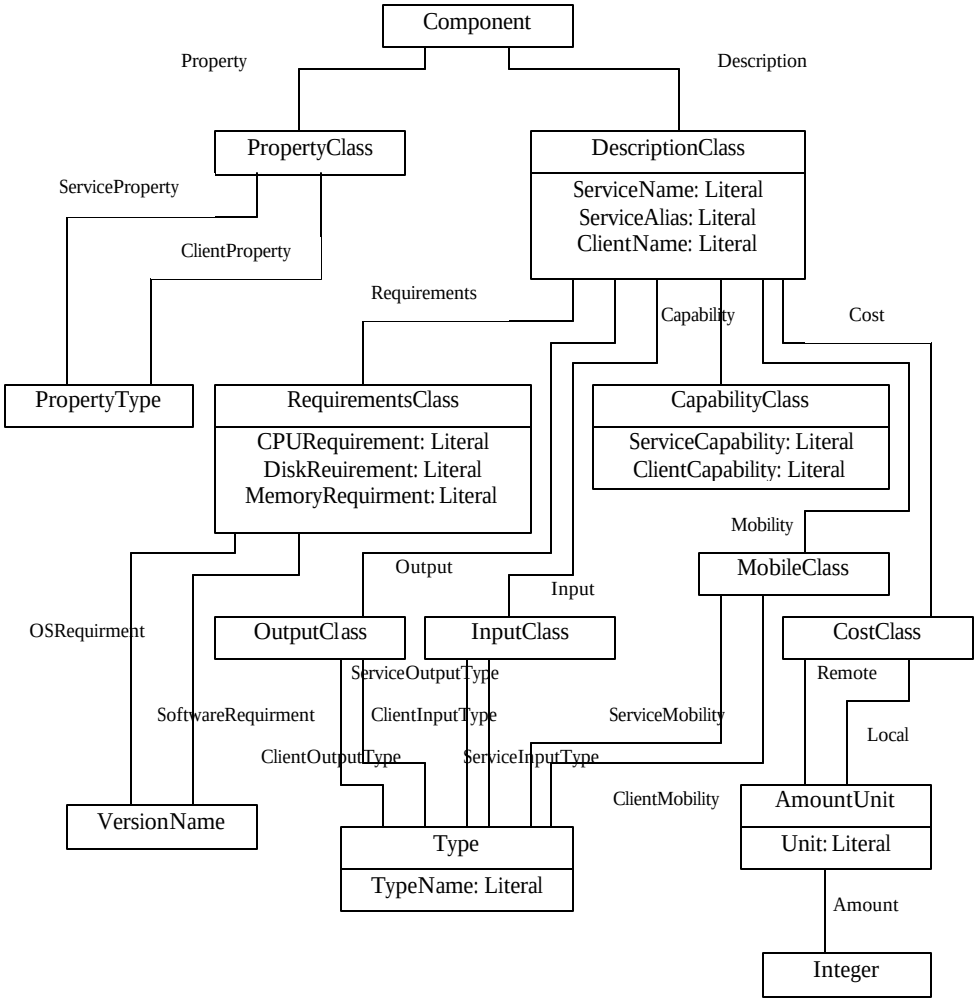
```
</rdf:Property>

<rdf:Property rdf:ID="Size">
    <rdfs:range rdf:resource="#AmountUnit" />
</rdf:Property>

<rdf:Property rdf:ID="Memory">
    <rdfs:range rdf:resource="#AmountUnit" />
</rdf:Property>

<rdf:Property rdf:ID="FileSystem">
    <rdfs:range rdf:resource="#AmountUnit" />
</rdf:Property>

<rdf:Property rdf:ID="OperatingSystem">
    <rdfs:range rdf:resource="#AmountUnit" />
</rdf:Property>

<rdf:Property rdf:ID="Software">
    <rdfs:range rdf:resource="#AmountUnit" />
</rdf:Property>

<rdfs:Class rdf:ID="AmountUnit">
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal" />
</rdfs:Class>

<rdf:Property rdf:ID="Amount">
    <rdfs:range rdf:resource="http://www.w3.org/1999/XMLSchema-datatypes#integer"
    />
    <rdfs:domain rdf:resource="#AmountUnit" />
</rdf:Property>

<rdf:Property rdf:ID="Unit">
    <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal" />
    <rdfs:domain rdf:resource="#AmountUnit" />
</rdf:Property>

<rdfs:Class rdf:ID="Type">
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal" />
</rdfs:Class>

<rdf:Property rdf:ID="TypeName">
    <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal" />
    <rdfs:domain rdf:resource="#Type" />
</rdf:Property>

<rdfs:Class rdf:ID="VersionName">
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal" />
```

</rdfs:Class>

</rdf:RDF>

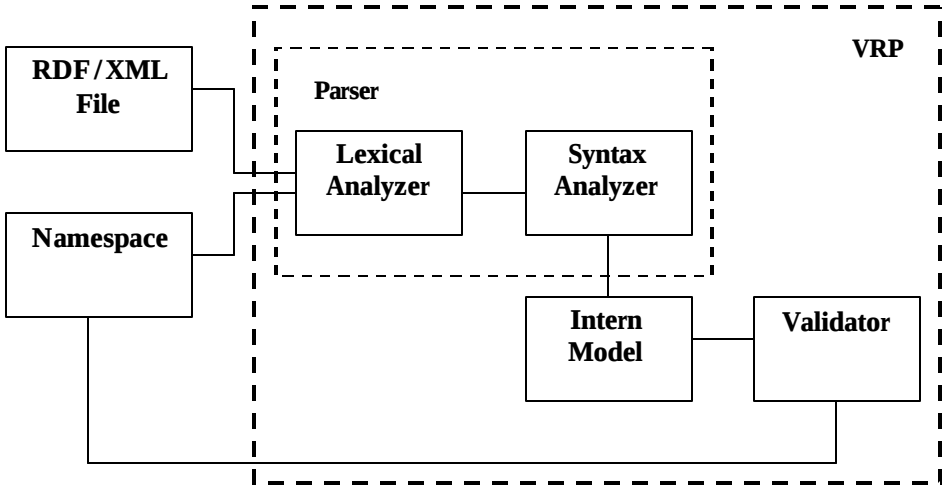
Appendix B: Hierarchy Diagram

This is a brief diagram for class hierarchy according to the previous RDF schema.



Appendix C: Overview of VRP

The ICS-FORTH Validating RDF Parser (VRP) contains a parser and a validator module. The parser analyses the statements of a given RDF/XML document according to the RDF/S specification. These statements are represented by a RDF object model implemented in Java™. The validator access the generated object model in order to validate the information against the RDF Schema constrains. Several output options are supported by VRP.



Validating RDF Parser System

Main Features

- ✧ 100% Java™
- ✧ need Java™ 1.2 or higher
- ✧ understands embedded RDF in HTML or XML
- ✧ full Unicode support
- ✧ based on compiler generator tools for Java CUP/JFlex
- ✧ provide API to easily integrate with other system