6-9-2006

An Indexation and Discovery Architecture for Semantic Web Services and its Application in Bioinformatics

Liyang Yu

AN INDEXATION AND DISCOVERY ARCHITECTURE FOR SEMANTIC WEB SERVICES AND ITS APPLICATION IN BIOINFORMATICS

by

Liyang Yu

Under the Direction of Rajshekhar Sunderraman

ABSTRACT

Recently much research effort has been devoted to the discovery of relevant Web services. It is widely recognized that adding semantics to service description is the solution to this challenge. Web services with explicit semantic annotation are called Semantic Web Services (SWS). This research proposes an indexation and discovery architecture for SWS, together with a prototype application in the area of bioinformatics. In this approach, a SWS repository is created and maintained by crawling both ontology-oriented UDDI registries and Web sites that hosting SWS. For a given service request, the proposed system invokes the matching algorithm and a candidate set is returned with different degree of matching considered. This approach can add more flexibility to the current industry standards by offering more choices to both the service requesters and publishers. Also, the prototype developed in this research shows the value can be added by using SWS in application areas such as bioinformatics.

INDEX WORDS: Web service standards, Semantic Web, Semantic Web services, OWL-S, Ontology, Indexation, Service discovery, Search engine, Web crawler, Bioinformatics applications

AN INDEXATION AND DISCOVERY ARCHITECTURE FOR SEMANTIC WEB SERVICES AND ITS APPLICATION IN BIOINFORMATICS

by

Liyang Yu

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of

Master of Science

in the College of Arts and Sciences

Georgia State University

2006

AN INDEXATION AND DISCOVERY ARCHITECTURE FOR SEMANTIC WEB SERVICES AND ITS APPLICATION IN BIOINFORMATICS

by

Liyang Yu

Major Professor: Rajshekhar Sunderraman
Yingshu Li
Yanqing Zhang

Electronic Version Approved:

Office of Graduate Studies
College of Arts and Sciences
Georgia State University
May 2006

Dedicated to my parents,
Hanting Yu and Zaiyun Du

ACKNOWLEDGEMENTS

My gratitude first goes to my adviser, Dr. Rajshekhar Sunderraman, for his intellectual support, encouragement and patience. During the course of this research, he provided me with clear directions, detailed instructions; and his understanding of this relatively new area also proves to be valuable. I would also like to mention that I enjoyed his course, “Database and Web”, a lot; it is my favorite course here in the Department of Computer in Georgia State University.

I would also like to thank my parents for their support and understanding, for all their worries about me as well. My mother keeps wondering why I have to go to school and take so many courses all the time. Well, I wanted to tell her that I am now done with schools and I enjoyed them very much.

I owe a special debt of gratitude to Ms. Jin Chen, for her encouragement, understanding and especially for her ever-lasting confidence in my ability to understand the area and continue the research. Quite often, she is the one who gets to know my progress first, listens to my boring talks first (and normally the only one), and suffers my endless complaints first and all by herself. And these days, I finally can let her rest and be happy for a while.

Finally, let me thank my fellow graduate students and developers in my work for being my friends, for participating in interesting discussions and for being patient enough to listen to my talks.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Web Services are considered as the core technology of e-Business platforms. The wide-spreading of Web services in the Intranets and in the near future in the whole Internet reveals the needs of sophisticated discovery mechanisms. This research is about finding the desired web services with high accuracy and efficiency. This chapter presents the motivation for this work by introducing the limitations of current standards for information and service discovery. The research questions are then introduced followed by the structure of this document.

1.1 Motivation

1.1.1 Finding Information on World Wide Web

World Wide Web contains virtually boundless information in the form of documents and one can use computers to search for these documents. For instance, using a common search engine, one can search the word “SOAP”. Unfortunately, one will find the results hardly helpful: there are listings for dish detergents, facial soaps, and even soap operas mixed into the results. Only after sifting through multiple listings and reading through the linked pages is one able to find information about the W3C’s SOAP specifications.

The reason of this situation is that search engines implement their search based on the core concept of "which documents contain the given key word" – as long as a given document contains the key word, it will be included in the candidate set that later presented to the user as the search result. It is then up to the user to read and interpret the result to extrapolate any useful information. To summarize this situation, computers can present users with information but they cannot understand what the information is well enough to display the data that is most relevant in a given circumstance.

To address this common difficulty with the World Wide Web, the Semantic Web vision was conceived by Tim Berners-Lee [1], the inventor of the World Wide Web. Calling it the next step in Web evolution, Berners-Lee defines the Semantic Web as "a web of data that can be processed directly and indirectly by machines." Therefore, semantic Web is about having data as well as documents on the Web so that machines can process, transform, assemble, and even act on the data in useful ways.

Let us still use the "SOAP" example to illustrate the basic idea. In the above "SOAP" example, because of the different semantic associations of the word "soap", the results the user receives are varied in relevance, and the user still has to do a detailed check to find the information he/she is looking for. However, if these semantic data is added to all these ordinary Web documents and with the help of these semantic metadata, machines can act as if they "understand" the information they are carrying.

Now, let us assume that the user is using a Semantic Web agent (a search agent who is capable of recognizing the newly added semantic metadata) to search the Web for "SOAP" where SOAP is a type of technology specification used in Web services. This time, the results of the search will be relevant. The returned results will not contain information such as dish detergents, etc. Based on the semantic information available for SOAP, the agent can also return a list of related technologies therefore it will become usefully clear to the user that WSDL, XML, and URI are all technologies related to SOAP.

This small example shows how the semantic Web can potentially change the way of finding relevant information from the World Wide Web. As a summary, in the semantic Web, data itself becomes part of the Web and is able to be processed independently of application, platform, or domain. This data is then used as semantic filter to decide which document should be included in the resulting set.

It is then interesting to explore the possibility of adding semantic information to traditional Web service descriptions and to understand how this added semantic information would change the way of discovering the desired Web services. This is the main motivation of this research and it is presented in great detail in the next section.

1.1.2 Discovering Traditional Web services

Web services are modular, self-describing, and self-contained applications that are accessible over the internet [2]. The core components of the Web services infrastructure are XML based standards like SOAP [3], WSDL (Web Services Description Language [4]) and UDDI (Universal Description Discovery and Integration [5]).

SOAP is the standard messaging protocol for Web services. SOAP messages consist of three parts: an envelop that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined data types, and a convention for representing remote procedure calls and responses.

WSDL is an XML format to describe Web services as collections of communication endpoints that can exchange certain messages. A complete WSDL service description provides two pieces of information: an application-level service description (or abstract interface), and the specific protocol-dependent details that users must follow to access the service at a specified concrete service endpoint.

The UDDI specifications offer users a unified and systematic way to find service providers through a centralized registry of services that is roughly equivalent to an automated online “phone directory” of Web services. UDDI provides two basic specifications that define a service registry’s structure and operation. One is a definition of the information to provide about each service and how to encode it and the other is a publishing and query API for the registry that describes how this information can be published and accessed.

In the space of discovery, UDDI is emerging as the main tool for Web service discovery. Ideally, for a desired Web service functionality, one should be able to use UDDI to locate a set of Web service candidates that are qualified for the request; and within this candidate set, UDDI should be able to suggest the single one service which has the highest QoS. However, the only discovery mechanism provided by UDDI is a keyword search on the names and features of businesses and services descriptions.

Unfortunately, keyword search fails to recognize the similarities and differences between the capabilities provided by Web services. Ultimately, UDDI is useful only to find information about known Web services and it completely fails as a general Web services discovery mechanism.

Given the fact that semantic Web offers new improvements to accomplish more efficient information retrieval, it is then possible to consider adding semantic information to the traditional Web services and exploring a new set of discovery methods to facilitate the discovery of Web services. This introduces the idea of semantic Web services.

1.1.3 Semantic Web Services and Ontology

As we discussed earlier, the current Web is just a collection of documents which are human readable but not machine processable. In order to remedy this disadvantage, the concept of semantic Web is proposed to add semantics to the Web to facilitate the information finding, extracting, representing, interpreting and maintaining. “The semantic Web is an extension of the current Web in which information is given well-defined meaning, better enabling computers and people to work in cooperation” [1].

The key reason why the current UDDI discovery mechanism fails is because the current Web services standards are not semantic-oriented and they are awkward for service discovery, invocation and composition. A seemingly obvious solution then is to combine the semantic Web with traditional Web services; this further creates the concept of semantic Web services. Web services with explicit semantic annotation are called Semantic Web Services (SWS) [6] – its vision is the application of semantic description for Web services in order to provide relevant criteria for their automated discovery.

The fundamental concept of semantic Web and semantic Web services is ontology, which provides the "well-defined meaning" to the information that is contained in the Web. "Ontology is a set of knowledge terms, including the vocabulary, the semantic interconnections, and some simple rules of inference and logic for some particular topic" [7]. The main benefit provided by ontology is that different parties over the internet now have "shared" definitions about certain key concepts.

For semantic Web services, ontology is again one of the core components: 1) using ontology brings user requirements and service advertisements to a common conceptual space; 2) using ontology also helps to apply reasoning mechanism to find a better service candidate. All these will be discussed and illustrated in this research in the later chapters. It will be shown that ontology based semantic Web services is the solution to the weakness of UDDI when discovering the desired services.

Ontology in industry practice is normally domain-specific so to limit the scope of the ontology. In this research, we are using bioinformatics as the example domain; therefore, a small ontology in this area is built to illustrate our contribution, this is also shown in the later chapters.

1.1.4 New Challenge: Semantic Web Service Discovery

After adding semantic information to Web services, the new challenge is how to take advantage of the semantic information to design a new discovery process. This is the main motivation for this research. A general discussion is presented in this section, the proposed architecture and methods will be presented in the following chapters.

Discovery is the process of finding Web services with a given capability. In general, this requires that Web services advertise their capabilities with a registry, and that requesting services query the registry for Web services with particular capabilities. Assuming an infrastructure that is based on a centralized registry, the role of this registry is both to store the advertisements and to perform a match between the request and the advertisements.

Assuming also that a domain specific ontology is built into the registry, this ontology is then used by both the service provider and requester. For the service provider, by mapping concepts in a service description to the concepts defined in the ontology, the semantics of the provided service is explicitly defined. For the service requester, the service request is also expressed using the concepts from the same ontology in the given domain. The final result is, by having both the service description and service request explicitly define their semantics, the results from the discovery process will be much more relevant than the results from simple keyword matching discovery. Therefore, the key to semantic discovery of Web services is having semantics in the description itself and then using semantic matching algorithms to find the required services.

Although the above basic idea is widely accepted in the research community, however, much still needs to be done. For instance, a domain-specific ontology, a registry/repository and a semantic agent have to be integrated together to implement the discovery process, different matching algorithms have to be studied. This serves as our main motivation for this research, and a formal problem statement is presented in the next section.

1.2 Problem Statement

This research will accomplish the following:

- propose an architecture for registration, indexation and discovery of semantic Web services, including the possible matching algorithms. The registration/indexation and discovery process should be based on the semantic matching instead of keyword searching as used in the traditional UDDI discovery mechanism;
- develop a prototyping system which implements the above proposed architecture using Bioinformatics as the application area to illustrate the interaction of different components in the

proposed architecture and how they work together to accomplish the goal of discovering the desired Web services based on a given service request.

1.3 The Organization of this Document

The rest of this thesis is organized as follows. Chapter 2 presents literature review to summarize the work that has been done toward the same direction, i.e., discovery of semantic Web services. A registration/indexation and discovery architecture is proposed in Chapter 3 and detailed discussion of main components, including the matching algorithm used in this architecture is also presented in the same chapter. Chapter 4 presents the proposed matching algorithm in more detail, and Chapter 5 summarizes the implementation techniques, including development tools, examples and other implementation details. The main contributions and future studies are summarized in Chapter 6.

CHAPTER 2

A BRIEF LITERATURE REVIEW

As stated in Chapter 1, to discover the desired Web services accurately and efficiently, current research trend is to add semantic information to the Web services framework (such as UDDI/WSDL) to facilitate the discovery and other capabilities such as automatic invocation and composition. This also includes the focus on designing different matching algorithms. Some related and seemingly popular approaches are briefly summarized in this Chapter.

2.1 Adding Semantics to WSDL/UDDI

Quite a few research works have been concentrating on directly adding the semantic descriptions into the WSDL and UDDI standards. The main benefit of this approach is that the existing traditional Web services infrastructure can be reused.

One such approach is suggested in [8]. More specifically, semantic information is added to both WSDL and UDDI, and semantic discovery algorithm is then used to find the desired services. This approach follows the current Web service standard framework by using the extensibility feature of WSDL and using UDDI data structure to represent grouping of operations with their inputs and outputs. For instance, each WSDL description may have a number of operations with different functionalities. In order to add semantics and to find relevant operations, these operations are mapped to concepts in appropriate DAML+OIL [9] ontology that depicts the functionality of operations. This allows users to search for operations based on ontological concepts. Also the message parts, i.e., input and output parameters of

operations that are defined in WSDL files using XML schema constructs, are mapped to their appropriate ontological concepts respectively.

As for the UDDI, semantic information is added by using the tModels. Four different tModels have been created and registered in [8]: the first tModel represents the ontology of concepts representing functionality of operations, the second and third represent the ontology of input and output concepts respectively. Finally, the fourth tModel represents the grouping of each operation with its inputs and outputs. The concepts that are represented by these tModels can be linked back to the concepts in the WSDL file, and all this semantic information can be used by the matching algorithm to find the appropriate candidate set.

Meteor-S [14] is a detailed extension of the work presented in [8]. It proposes a framework for adding semantics directly to existing Web services standards, such as WSDL and UDDI. It allows users to semantically annotate their WSDL and UDDI descriptions of their Web services with DAML and publish these descriptions in their enhanced UDDI. Their matching algorithm extended the work presented in [10] in two ways. First they extend the subsumption based matching mechanism by adding information retrieval techniques to find similarity between concepts when it is not explicitly stated in the ontologies, and secondly they added a mechanism to match on preconditions and effects of service descriptions.

Another similar solution is discussed in [6]. Instead of using DAML+OIL, OWL-S [11] is used to describe the semantics of a given Web service. OWL-S "grounding" is proposed to take care of the mapping from the concepts that describe the inputs and outputs of the services to the inputs and outputs of the corresponding operations in the WSDL specifications. To facilitate the discovery process, OWL-S service profile is mapped into UDDI registry in the way quite similar to the approach that is used in [10] as discussed below. This added semantic layer in UDDI makes it not only a registry for storing the advertisements of Web services, but also a possible location to perform semantic match between the request and the service capabilities.

A relatively simpler approach is proposed in [10], this solution does not involve the change to WSDL. In this work, the discovery of Web services is implemented in two steps. The first step is to use DAML-S upper ontology to describe the semantic representation of the given Web service. More specifically, "service profile" of DAML-S upper ontology is used to describe the capabilities of the service in terms of inputs, outputs, preconditions and effects. The given service is considered to be fully defined after the description of these four aspects of the services is completed.

The second step is to translate the DAML-S representations into UDDI representation, i.e., to map the description represented by DAML-S upper ontology into UDDI registry. The key in this step is the UDDI tModel, which is a data structure provided by UDDI and can be used to specify information about the services. The actual mapping is done by adding a set of 15 UDDI tModels, one for each attributes of the DAML-S profile representation. One of the tModels, the DAML-S tModel, has a special meaning: it states that the service advertised has a DAML-S service representation. A simple matching engine is also proposed in [10]. On top of the UDDI searching APIs, this added matching engine can find the Web services based on their semantic representations which are embedded in the UDDI registry as described above.

Quite a few other research efforts are also based on the idea of combining DAML-S/OWL-S and WSDL/UDDI. For example, the work reported in [12] combines OWL-S and UDDI without changing WSDL; this is very similar to the work in [10]. However, it also considers the issue of scalability and performance, a preliminary result on performance issue is reported in [12]. The conclusion is, comparing the performance of the OWL-S/UDDI registry and a UDDI registry, adding an OWL-S layer and its corresponding matching component does not hinder the performance and scalability of a UDDI registry.

Another example for a semantic UDDI registry is presented in [13]. This work is based on [10] but provides an enhancement to the semantic search mechanism presented in [10] in several ways. First, it extends the UDDI inquiry API by enabling users to specify semantic inquiries based on Web services capabilities, secondly it enhances the matching algorithm with a planning functionality, which is capable of satisfying users requests by composing two or more service descriptions.

A somewhat different approach without making changes to WSDL/UDDI is the work presented in [15]. Internet Reasoning Service (IRS-II) introduced in this work shows how to support the publication, location, composition and execution of heterogeneous semantic rich web services. It uses UPML(Unified Problem Solving Method Developments Language) framework [16] for the specification of reusability in knowledge-based systems by defining how we can build elementary components and how these components can be integrated into one whole system. The IRS-II approach enables applications to semantically describe and execute Web services. It supports capability-driven service invocation (e.g. find a service that can solve problem X) because of the explicit separation of task specifications (the problems which need to be solved), method specifications (the ways to solve problems), and domain models (the context in which these problems need to be solved).

2.2 Designing Architectures for Semantic Web Services

As discussed in Section 2.1, much of the work on Web services discovery is based on modifying/extending a centralized registries such as UDDI repository. Although centralized registries are effective since they guarantee the discovery of services that are registered, they suffer from traditional problems of centralized systems as bottlenecks and single points of failure. Also, replicating the UDDI data is not a scalable approach. Therefore, some of the research effort concentrates on alternative architecture for the discovery of semantic Web services.

One such example is given by the work in [17]. They argue that the trend in integrating UDDI feature into general purpose enterprise registries results in rapid increase of private registries and limits the use of public registries. They also state that from discovery perspective, it is impractical to replicate these private registries in the public counterparts. They propose to deal with this problem by creating a virtual global registry by connecting all private registries in a P2P network. The support for semantic based service discovery is done by using DAML-S service descriptions and matchmaking.

A similar but extended work is presented in [18]. In this work, multiple public/private registries are grouped into registry federation. More precisely, a registry federation is a collection of autonomous but cooperating Web service registries. The goal of a federation can be forming a registry community serving either a business domain or forming a market place of registries with similar but competing services. The publication and discovery of semantic Web services requires efficient access to these federations, and this efficient access is implemented by storing semantic metadata in the form of Extended Registries Ontology (XTRO), which successfully captures the relationships among these registries and also categorize the registries on the basis of business domes.

Another federated architecture is presented in [19]. The proposed federated architecture supports QoS based discovery of services. It has a notion of UX ("UDDI Extension") server that performs federated discovery on behalf of a user request and aggregates results before sending them back to the requester. The paper discusses different ways of maintaining links between the servers and how query is propagated. It also envisions linking UX servers across different domains. It points out that improvements could be made using semantic descriptions and matchmaking. In the work presented in [18], this idea is pushed further, the emphasis is rather on utilizing UDDI data structure to store semantic description of a service for better service matchmaking, to establish a federation for carrying out discovery process in multiple registries and in exploiting an ontology for improving the registry selection mechanism for Web service publication and discovery.

Another P2P example is given in [20] which uses a P2P architecture to facilitate the publication and discovery of semantic Web services. For a service provider, the semantic description of the service is coded in such a way that it also identifies the server in which this service will be published; it is the mechanism for the dissemination of service description in the P2P overlay of the location servers. For a client, the semantic description of the request is also coded to identify the server that probably contains the discovery information for the requested service; it is the research mechanism of service description in the P2P overlay of the location servers. Once the specific server is located, the rest is done by the proposed matching algorithm.

A recent architecture for registration and discovery of the semantic Web services is proposed in [21]. This architecture includes a web crawler which will not only visit the Web sites that publish semantic Web services and collect all these service descriptions into its only registry, but will also visit public/private UDDI registries to search for service advertisements which also have semantic augmentation. Once these service advertisements are discovered, they are also gathered into the registry that is maintained by this architecture. A semantic matchmaking engine is provided so for the service requests, a candidate set will be proposed. To select the "best" candidate from this candidate set, fuzzy neural networks with Genetic Algorithm is used.

The work in this research is an extension to the work presented in [21] with an application example in the area of Bioinformatics. The overall architecture, the implementation issues, running examples and summary of contribution are presented in details in the next few chapters.

2.3 Matching Algorithms

Clearly, the discovery of semantic Web services depends on the semantic match between a declarative description of the service being sought, and a description of the service being offered. Therefore, the study of different matching algorithms attracts considerable amount of research effort. This section presents a brief review of some of the efforts.

The work in [22] considers a setting where the service capabilities are described using DAML-S upper ontology and the semantic matching is performed between advertisements and requests. More specifically, DAML-S Service Profiles describe service requests as well as service provided. A request consists of a description of a hypothetical service that performs a task needed by the requester. Requests are sent to registries of Web services that match them against the profiles advertised by other services to identify which services provide the best match. A matching is successful when all the outputs of the request are matched by the outputs of the advertisement, and all the inputs of the advertisement are matched by the inputs of the request. The "degree" of matching is decided by whether the matching

between concepts is exact or subsume. The main advantage of this algorithm is its simplicity and therefore easy to be implemented.

Since DAML-S has been updated and renamed to OWL-S, adapting the evolution of DAML+OIL to OWL [11], some researchers have developed matching algorithms based on OWL-S. One of these examples is given in [23]. A service provider describes his advertised services in an OWL-S compliant ontology and a service requester queries for services with an OWL-S ontology expressing his requirements. The proposed algorithm is divided into four stages: (a). the matching of inputs, (b) the matching of outputs, (c) the matching of service category, and (d) user-defined constraints or functionality that can be used to guide the matching process. In this algorithm, the relationship between concepts is also considered: concept B is subsumed by concept A, meaning that A denotes a more general concept than B. Although this same relationship is also considered in [22], the service category and user-defined constraints are not included in the algorithm proposed in [22].

As a summary, all these matching algorithms concentrate on the service profile and its inputs and outputs for determining matches between requests and advertisements. This decision is shared by many other matching algorithms that are proposed in related research work, see [20] and [24] for other examples. Besides the above seemingly popular approach that is based on inputs and outputs, a different approach is found in [25]: a match maker is designed that covers only the service model of DAML-S. As pointed out in [26], this service model is not primarily provided to express requirements for finding matches with advertisements.

2.4 Discussion

This chapter is not intended to be a comprehensive review of the current research efforts; rather, it only summarizes the seemingly popular results in the area of Web service discovery. Based on this brief literature review, we have established the following observations:

- It is commonly accepted by the academic community that the current de facto Web service standards, especially UDDI registries, it is not powerful enough to support dynamic discovery of desired Web services;
- The solution to the above challenge is to add semantics to the descriptions of the given Web services, i.e., to change the traditional Web services into semantic Web services. A popular way to implement this is to add semantic information to WSDL and/or UDDI;
- Since the main solution is to add semantics to WSDL/UDDI, the resulting architecture is still the centralized UDDI registries. However, other alternative, such as P2P structure (UDDI federations) has been proposed;
- Different matching algorithms have been proposed to facilitate the discovery of semantic Web services. Matchmaking algorithms is normally proposed in the context where 1) a domain specific ontology is created to express semantic capabilities of services and these services are described with their inputs and outputs; 2) for service discovery, clients and providers should use the same ontology to describe requests and services.

The research work in this thesis is mainly based on the initial work presented in [21]. More specifically, we propose an architecture that will facilitate the registration/indexation and discovery of semantic Web services. A Web crawler in this framework will access specific service ontology-oriented UDDI registries to fetch the service profiles, translate them into the format supported by the repository in our framework; the crawler will also crawl the semantic Web sites which hosting the specific ontology based semantic Web services directly to get the service profiles, and save them in our repository. To some extend, this repository is similar to the index database that is maintained by a traditional keyword based search engine. Furthermore, this crawler is controlled by a “broker”, who will periodically start the crawler so the repository is being reasonably up-to-date. Besides this registration and indexation process, the broker will also frequently receive service requests and therefore start the matching engine to find a

"candidate set" of Semantic Web services and return this set to the requestor. This discovery process is implemented by a proposed matching algorithm between advertisements and requests described in OWL-S that recognizes various degrees of matching to consider different potentials for maximizing the quality of service (QoS).

To illustrate this proposed architecture and the matching algorithm, an example in the field of bioinformatics is implemented. In this example, some frequently used services, such as protein structure prediction and sequence alignment, are published as semantic Web services, and after receiving a service request, the broker is able to find the related candidate set and return it to the requestor.

The following chapters will present this research work and implementation example in detail. We believe the proposed approach in this research can add more flexibility to the current industry standards and also offer more choices to both the service requestors and publishers. The main contribution from this research is also summarized in the later chapters.

CHAPTER 3

AN INDEXATION AND DISCOVERY ARCHITECTURE FOR SEMANTIC WEB SERVICES

In this Chapter, the proposed architecture for indexation and discovery of semantic Web services is presented. The details of each component and the relationship between each component in this architecture are also discussed here. The key benefits of this architecture are then summarized.

3.1 Discussion

In the Web service architecture description presented by the W3C [27], three main discovery scenarios are identified: a centralized registry, a peer-to-peer (P2P) scenario and an index registry scenario.

Perhaps the most popular structure is a registry structure. In such a scenario, a centralized "yellow book" stores service descriptions which are submitted by service providers. An existing example of such a repository is obviously UDDI [5]. To add semantics to Web service descriptions, UDDI can be extended to be compatible with DAML-S or OWL-S. And as stated in Chapter 2, the semantic information is added into UDDI by creating new tModels. When a service request is submitted, the UDDI server will suggest the candidate Web services by matching the requirements with the descriptions in the registry based on some specific domain ontology. Examples of this approach can be found in Chapter 2 and this scenario is outlined in Figure 1.

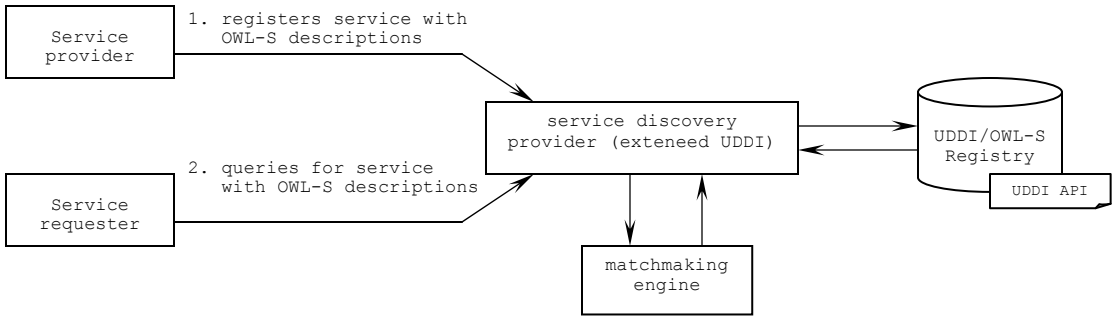

Fig. 1. Service discovery using a centralized registry

A P2P scenario is just the opposite of the above registry scenario: each Web service is being discovered dynamically, a service requester queries other nodes in its network or some specific network domain to find and identify the candidates without the existence of a centralized registry or index. As summarized in [23], this architecture means more updated service descriptions, but the matching effort could be more complex. Also, a key issue is where the matchmaking process is performed: it can be performed either on a centralized server or by individual clients. As an advantage of the server-sided scenario, the implementation of a client (acting as the service requester), can be kept very simple. Thus, the effort for finding services on the client side is very low. This issue is a well known important factor when selecting a server-oriented architecture in general. For instance, a service requester may want to use his/her own matching algorithm instead of the implementation on a central server. These custom modules can define constraints that must be satisfied by the OWL-S description of the advertised service. Only with the client-sided execution of such modules the personalization of the matching process is possible.

Another approach is the index scenario. The index is built by the so-called crawlers or robots [28] that browse the Web automatically. The difference between the index and the registry approach is that service providers control what information is put into the registry, whereas the index collects information on its own, in most cases automatically. The benefit is that the crawlers/robots can decide what information needs to be gathered based upon the needs of the possible matching algorithms. Also,

compared to the centralized registry such as UDDI, the index approach tends to be more updated since the crawlers/robots can walk thought the Web periodically to update the index.

In this research, an indexation/repository structure is selected to build the registry. More specifically, an index of semantic Web service description is created and maintained by a crawler which visits not only the general Web sites to collect published semantic Web service descriptions, but also the public/private UDDI registries. A matchmaking engine is then built to represent a single point of contact between the repository and a service requester; it performs the matchmaking of service advertisements and requests. Since this engine is located on the server and the process of matchmaking is performed on the server side, there is very little effort from the clients.

An important feature of the crawler that is obvious from the above description is the comprehensive coverage: the crawler will not only walk thought the Web to collect all the published Web service descriptions, but will also visit public/private UDDI registries to search for Web service descriptions that are semantically enhanced. This means that the service requester using the proposed system will have a better chance to find the desired service: he is not only searching the UDDI registries, but also the whole Web for the potential candidates. The proposed architecture is described in the following section and more discussion of its benefits is presented at the end of this chapter.

3.2 Architecture of Indexation and Discovery of Semantic Web Services

Based on the above discussion, the proposed architecture for indexation and discovery of Semantic Web services is given in Figure 2. It is built upon specific domain ontology (in this study, we use bioinformatics as the specific domain, see the Chapter 5 about implementation). This architecture has the following major components: (a) web crawler; (b) semantic Web service (SWS) repository; (c) matchmaking engine; (d) broker; (e) domain specific ontology. Each of these main components is described next.

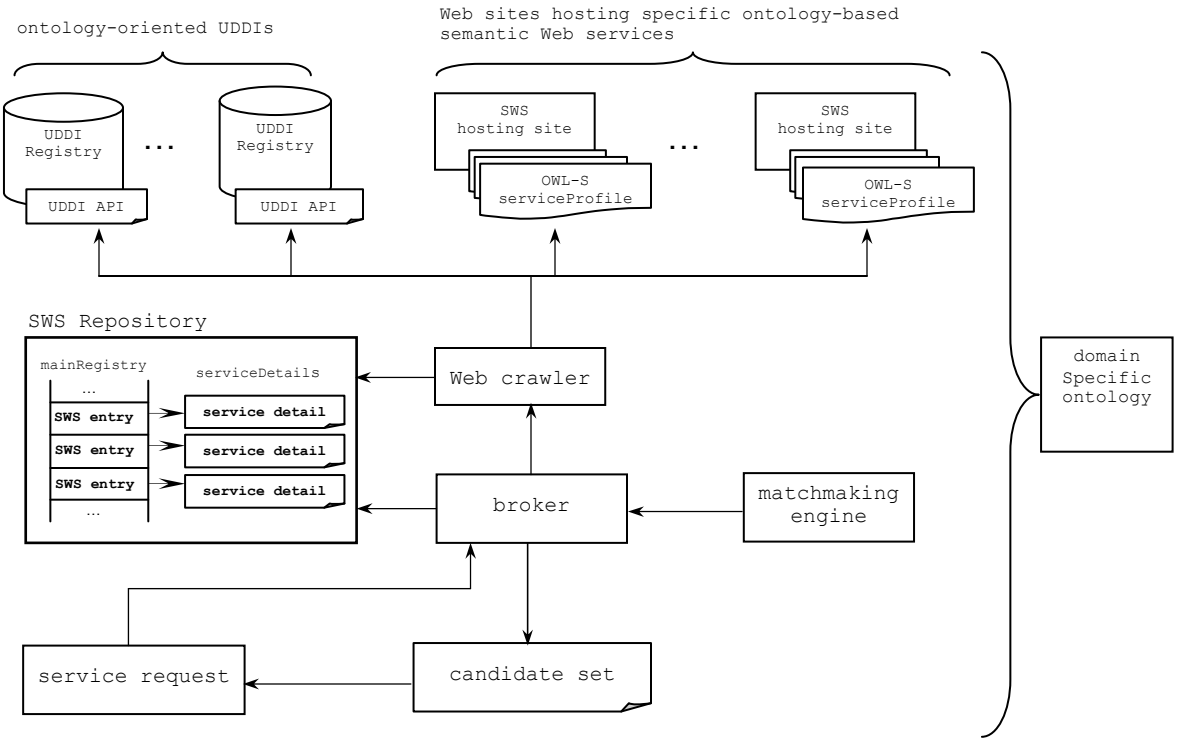

Fig. 2. Architecture of the semantic Web service indexation and discovery

3.2.1 Web Crawler

As shown in Figure 2, the web crawler has two main tasks:

1. Accessing the public/private specific service ontology-oriented UDDI registries using UDDI query API to fetch the service profiles, transforming them into the format supported by the proposed repository, and storing them into the repository using the published API of the repository;
2. Crawling the semantic Web sites hosting the specific ontology based semantic Web services directly to get the service profiles, transforming them into the format supported by the repository, and storing them into repository using the published API for the repository.

The reason for proposing such a Web crawler is also quite obvious. Under the current semantic Web service environment, UDDI registry must be extended to be ontology-compatible which supports

semantic matching of semantic Web services' capabilities. One such approach, as discussed in Chapter 2, is to map the OWL-S service profiles into current UDDI registry's data structure. In order to include this popular solution and to reuse and extend the existing infrastructure, the Web crawler in the proposed architecture has to be able to visit these public/private UDDI registries to collect the semantic descriptions.

In order to offer more flexibility to both service providers and service requesters, the Web crawler, besides visiting the UDDI registries, is designed to be able to craw the Web, collecting the semantic Web service descriptions directly from the hosting Web sites. This directly implies much more flexibility to the service providers: they can either publish the service profiles of semantic Web services into the public/private specific service ontology-oriented UDDI registries or directly on their hosting Web sites. For the service requesters, they now have a better chance to find the desired service: using the proposed system, the service requesters are not only searching the UDDI registries but also "all" the Web sites hosting semantic Web services.

After visiting the related sites, the crawler will transform the gathered semantic descriptions into the format supported by the repository and store them into the repository. When a service request arrives, the broker will visit this repository and activate the matchmaking engine to recommend a list of candidates that may satisfy the request.

In this research effort, we are able to fetch quite a few web sites that contain semantic Web service profiles and therefore the crawler is able to collect these profiles into the repository, however, we are not able to find any semantic profiles in the UDDI registries except a single testing example. For details of the implementation work, see Chapter 4.

3.2.2 Semantic Web Service Repository

The specific ontology based semantic Web service repository (SWS repository) is responsible for storing the service profiles of semantic Web services. As described in Figure 2, its main purpose is to

"remember" all the semantic Web service descriptions that are collected by the crawler. The criteria used to design the structure of the SWS repository can be summarized as follows:

- The description data stored in this repository should be detailed enough that when a service request arrives, there is enough information for the matchmaking engine to construct the candidate set;
- The service requester should be able to invoke the selected service based on the information stored in the repository.

Based on the above considerations, the repository can be implemented using a DBMS, such as ORACLE and it has two main tables. The first table is shown in Table 1.

Table 1. `mainRegistry` table in SWS repository

```
SQL> describe mainregistry;
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------------
 SERVICEURL                                NOT NULL VARCHAR2(128)
 ONTOLOGYURL                                        VARCHAR2(128)
 SERVICENAME                                        VARCHAR2(128)
 CONTACTEMAIL                                       VARCHAR2(128)
 UDDIENTRY                                          VARCHAR2(256)

SQL>
```

The detailed description of each field in Table 1 is summarized in Table 2.

Table 2. Detailed descriptions of `mainRegistry` table

Field name	Description
`serviceURL`	Key of the `mainRegistry` table. If this description (the current record) is added by the crawler as a result of

	searching the UDDI registry, this field will be the standard UDDI `serviceKey` value which can uniquely identifies the give Web service in the UDDI registry (therefore, when this service is returned as a candidate to the service requester, this field can be used to easily retrieve the service description from the UDDI registry). If this record is added by the crawler as a result of visiting a Web site which directly hosts a semantic Web service (i.e., this description is obtained directly from the publisher's site), this field will be the URL of the OWL-S file describing the Web service.
`ontologyURL`	The URL of the domain specific ontology file.
`serviceName`	The name of the Web service.
`contactEmail`	The Email to contact the provider of this service.
`UDDIEntry`	In the case where this record is added by the crawler as a result of visiting a Web site which directly hosts a semantic Web service, it can be true that the service provider elects to not only publish the service on his/her Web site, but also enter the service description into the UDDI registry. If this is the case, this field will then serve as a "pointer" which links this semantic description (the OWL file) to the service details in the UDDI registry, i.e., the `serviceKey` value used in the UDDI standards. If the service provider does not offer an entry in the UDDI registry, then this field can point to the WSDL file that is associated with the given Web service.

The second table provides more details about a given Web service description, its structure is given in Table 3.

Table 3. serviceDetail table in SWS repository

```
SQL> describe serviceDetail;
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------------
 SERVICEURL                                NOT NULL VARCHAR2(128)
 PARAMETERTYPE                                      VARCHAR2(16)
 PARAMETERCLASS                                     VARCHAR2(32)

SQL>
```

Again, the detailed description of each field in the `serviceDetail` table is presented in Table 4.

Table 4. detailed descriptions of `serviceDetail` table

Field name	Description
`serviceURL`	Foreign key which links back to `mainRegistry` table
`parameterType`	Identifies whether this parameter is input or output parameter
`parameterClass`	Identifies the concept/class that is associated with this parameter. For instance, the concept/class of a given input parameter could be `ProteinStructure`, and clearly, this concept/class has to be defined in the domain specific ontology.

Clearly, `mainRegistry` and `serviceDetail` together will record a complete description of a given Web service. The matchmaking engine will use the description in these two tables when the matchmaking is implemented. For more details and examples, refer to the later chapters.

3.3.3 Matchmaking Engine

Based on the previous discussion, the proposed architecture can be viewed as a Web wide infrastructure for semantic Web service descriptions supported by a SWS repository that functions as a directory. This directory records advertisements of services that come on line or registered in UDDI registries, and supports the searching of services that provide a set of requested functionalities.

The matchmaking engine is the core component for the search of the semantic Web services. Its function is to use the built-in algorithm to find the potential Web service candidates based on a given service request. The followings are the main considerations when designing the engine:

- It has direct access to the SWS repository;
- It has direct access to the domain specific ontology;
- It has the ability to make inferences based on the given ontology;
- Its matching algorithm(s) can be modified, extended, and even “replaced”.

More specifically, for a given service request, the engine will read in the detailed information of the request, access the SWS repository, use the algorithm to analyze each entry in the repository and

decide its qualification based on the algorithm. The matchmaking engine has direct access to the domain-specific ontology that is used by both the service providers and service requesters.

It is clear that in the proposed design, the matching process is implemented on the server side, i.e., the effort of finding the candidate services is very low on the service requester (the client) side. This has the advantage of an extensible architecture since the matching algorithms in the matchmaking engine can be added, modified, updated and even streamlined so a better candidate set can be proposed. Also, different QoS can be implemented to improve the matching quality. This later topic is discussed more in detail as given in [21].

3.2.4 SWS Broker

The semantic Web service broker is the key component in the proposed architecture, and it has the following responsibilities:

- Creates and maintains the SWS repository by managing the Web crawler. When informed by the broker, the Web crawler will walk thought the Web sites and public/private UDDIs to update the repository periodically;
- Provides a single point of contact between the user and the system: users' service requests are received by the broker, and the broker will use the SWS matchmaking engine to access the repository and provide a candidate set of potential Web services that may satisfy the request, and return the list to the user.

Other research efforts (for instance, [21]) may use the name "intelligent agent", "communication server", "inquiry server", etc. For the proposed architecture, the SWS broker assumes all these responsibilities for the system. It is therefore the component which logically connects all the other components, such as the crawler, the SWS repository, the matchmaking engine and the domain specific ontology, to work together for the discovery of relevant Web services.

3.2.5 Domain Specific Ontology

As discussed before, the fundamental concept of semantic Web and semantic Web services is ontology, which provides the "well-defined meaning" to the information that is contained in the Web. For semantic Web services specifically, the terms (input, output, precondition, effect, for instance) that are used in both the advertisements and requests have to be defined in an ontology, and this ontology can be viewed as the "physical storage" of the semantics for each term.

The ontology component that is included in the proposed architecture is the single component that has to be accessible to all other components in the system. For instance, the crawler needs the ontology to "understand" a service profile and therefore correctly stores it into the SWS repository, the matchmaking engine needs the ontology to make inferences about the qualification of a given candidate. The main benefit provided by ontology is that different parties on the Web now have "shared" definitions and semantics about certain key concepts/classes; it brings user requirements and service advertisements to a common conceptual space. Even more important is the fact that using ontology also helps to apply reasoning mechanism to find a better service candidate (this will become clearer in later chapters).

Ontology in industry practice is normally domain-specific so to limit the scope of the ontology. Therefore, in the proposed structure, the ontology component has to be domain specific also. In this research, bioinformatics is used as the example domain; a small ontology in this area is built to illustrate the implementation details of the proposed system.

3.3 Benefits of the Proposed Architecture

The detailed discussion of the proposed architecture is presented in the earlier sections. In this section, we briefly summarize the benefits of this structure as follows:

1. More comprehensive coverage.

The Web crawler component visits not only the public/private UDDI registries, but also the Web to find all the Web sites hosting semantic Web services. This implies the directory in the proposed architecture has more coverage and therefore a search conducted against this directory can provide a better chance of finding the desired service. On the other hand, the current popular solution is to add semantic information into WSDL/UDDI (see Chapter 2 for details), in this solution, the matchmaking process is implemented solely in the scope of the UDDI registries. If a Web services provider decided not to register the service into UDDI registry but only publish it in his/her own Web site, this searching algorithm would miss this candidate for sure.

2. More flexibility to service publisher.

Compared to the UDDI registry structure, the proposed structure offers more flexibility to the service publisher. Besides the option to register the service into UDDI registry (and insert relevant UDDI tModels to add the semantic descriptions), the publisher can also elect to publish the service profiles on their Web sites. As long as these profiles are created by following the current standards, such as OWL-S, the crawler can successfully pick them up and store them into the index table (directory). Meanwhile, this also means more flexibility to service requestor: as argued above, querying the broker in our framework is equivalent to querying both UDDI registries and publishers' Web sites as the same time; the returned candidate set is more complete.

3. More up-to-date repository.

A centralized UDDI registry always suffers form the possibility of being obsolete, and keeping an UDDI registry up-to-date has to be done by careful planning and tedious programming work using its publish APIs. On the other hand, the service descriptions stored in the SWS repository in this structure can be easily kept to be up-to-date given the fact that the crawler can periodically re-do its visiting work.

4. More transparency to the client.

In this structure, the broker is able to hide the complexity of the matching algorithms. When a particular service is requested, the effort on the client side can be minimized. However, in the P2P case, the client is sometimes forced to have its own matching algorithms.

CHAPTER 4

MATCHMAKING ALGORITHM

4.1 Basic Assumptions

Matchmaking algorithms is the key to the discovery process of semantic Web services. Much research effort has been attracted in this area; examples of proposed matching algorithms can be found in the discussion of Chapter 2. As a summary, matching algorithms are often proposed with the following assumptions:

- a domain specific ontology is created to express semantic capabilities of services and these services are described with their inputs and outputs;
- for service discovery, clients and providers should use the same ontology to describe requests and services.

The matching algorithm that is developed in this research follows these same assumptions and concentrates on the service profile and its inputs and outputs for determining the degree of matching between requests and advertisements. This decision is shared by many other matching algorithms proposed in related research work, see [8], [20], [22], [23] and [24] for examples.

In this research, we also assume that the ontology under consideration is written in OWL and service providers are assumed to describe the advertised services using OWL-S upper ontology for service description, the same assumption also holds for service requests, i.e., service requests are also

expressed using OWL-S. These are reasonable assumptions because, as pointed out in [11], OWL and OWL-S are becoming the standards for the semantic Web service descriptions.

4.2 Description of the Matchmaking Algorithm

In this section, we describe the proposed matchmaking algorithm. The basic assumptions of this algorithm are summarized in Section 4.1. Since both the service requests and service advertisements are expressed in OWL-S (based on the shared ontology), in this section, the word "input" and "output" both refer to and are equivalent to the concepts/classes that are defined in the shared domain specific ontology. Let us start by introducing some notations which will then make the description of the algorithm easier.

- I_R: the set of all the concepts that are used as inputs of the desired Web service, expressed in the service request file provided by the service requester;
- I_P: the set of all the concepts that are used as inputs of the given Web service, expressed in the service description file presented by service provider;
- O_R: the set of all the concepts that are used as outputs of the desired Web service, expressed in the service request file provided by the service requester;
- O_P: the set of all the concepts that are used as outputs of the given Web service, expressed in the service description file presented by service provider;
- Ω: the set of all the concepts/classes that are defined in the domain-specific ontology.

Given the fact that both the service requests and service advertisements are expressed in OWL-S using a shared domain specific ontology, every input and output in these description files can be mapped to one and only one class/concept, denoted by C, defined in the domain specific ontology. The following can be used to formally express this relationship:

define $C=C(e)$, such that $\forall e \in (I_R \cup I_P \cup O_R \cup O_P)$, $C \in \Omega$.

We can then define the following mapping functions for the input sets:

$f_{input\text{-}exact}$: a 1-1 mapping from $I_R \rightarrow I_P$: $\forall e_{iR} \in I_R$, $\exists e_{iP} \in I_P$, such that $C(e_{iR}) \equiv C(e_{iP})$; "≡" means the left-hand-side (LHS) concept is equivalent to the right-hand-side (RHS) concept. Clearly, this implies that $|I_R| = |I_P|$, i.e., the number of input parameters required by the desired Web service requested by the server requester is equal to the number of input parameters required by the Web service provided by the service publisher.

$f_{input\text{-}L1}$: a 1-1 mapping from $I_R \rightarrow I_P$: $\forall e_{iR} \in I_R$, $\exists e_{iP} \in I_P$, such that either one of the following relation holds: $C(e_{iR}) \equiv C(e_{iP})$, or $C(e_{iR}) < C(e_{iP})$; and there exists at least one pair of $e_{iR} \in I_R$, $e_{iP} \in I_P$ such that the < relation holds. "<" means the LHS concept is a sub-concept or sub-class of the RHS concept. Clearly, this implies that $|I_R| = |I_P|$;

$f_{input\text{-}L2}$: a 1-1 mapping from $I_R \rightarrow I_P$: $\forall e_{iR} \in I_R$, $\exists e_{iP} \in I_P$, such that one of the following relation holds: $C(e_{iR}) \equiv C(e_{iP})$ or, $C(e_{iR}) < C(e_{iP})$ or, $C(e_{iR}) > C(e_{iP})$; and there exists at least one pair of $e_{iR} \in I_R$, $e_{iP} \in I_P$ such that the > relation holds. ">" means the LHS concept is a super concept or super class of the RHS concept. Clearly, this implies that $|I_R| = |I_P|$.

These 3 mapping functions are defined to test the matching between the provided inputs and the desired inputs. First of all, notice that each one of these testing function requires the condition $|I_R| = |I_P|$. Intuitively, this condition says that if the service requester would like to provide 4 input parameters to

accomplish some functionality, then in order for a given Web service to become a potential candidate, it should at least be able to accept exactly 4 input parameters. Clearly, this is a fundamental condition that every candidate should satisfy when the matching of the inputs is being tested.

Now, for a given candidate Web service, assume the number of its required input parameters is equal to the number of input parameters that the service requester is willing to provide (i.e., the above condition is met), and furthermore, if every input parameter's concept/class from the input set that the service requester is providing can find its exact matching counterpart in the given candidate's required input parameter set, one can comfortably declare that the input parameters required by the provider are exactly matching the input parameters offered by the requester. This perfect matching situation is captured by the mapping function $f_{input\text{-}exact}$.

Sometimes, the exact matching cannot be achieved but an acceptable matching is still possible. One such situation is where some of the input concepts provided by the service requester cannot find their exact counterpart classes but can instead locate their super-concepts (super-classes) in the input parameter set required by a given candidate. For instance, the service requester is willing to provide one input concept "SUV", and one candidate can accept "Vehicle" as input, the provided concept is a sub-class of the required concept (therefore, the provided concept has "more" information than that is actually needed), this given Web service can still be submitted as a candidate since the service requester in fact can provide more information to the service provider. This matching relationship is defined by $f_{input\text{-}L1}$ where `L1` stands for "`Level-1`" matching. Clearly exact matching is the perfect matching and more desirable than a "`Level-1`" matching.

The last level of matching considered in this matching algorithm is the "`Level-2`" matching denoted by $f_{input\text{-}L2}$. This matching happens when some of the input concepts provided by the service requester can find neither their exact counterpart classes nor their super-class counterparts (the `Level-1` matching scenario), but can instead locate their sub-concepts (sub-classes) in the input parameter set required by a given candidate. Use the same example as above, the service requester is willing to provide

one input concept "Vehicle", and the current candidate requires "SUV" as input class, the provided concept is a super-class of the required concept. In this situation, the provided concept has "less" information than the concept that is required by the service provider. However, it could be true that this "extra" information encapsulated in the sub-class is never really needed by the service, in which case this given Web service can again be presented as a candidate. In reality, it is very difficult to obtain the information whether this is the case or not, therefore, "`Level-2`" matching candidates are presented to the users as references an should always be used with care. Clearly, "`Level-2`" matching is less desirable than "`Level-1`" matching.

Similarly, to test the matching among the output concepts, the following mapping functions can be defined:

$f_{output\text{-}exact}$: a 1-1 mapping from $O_R \rightarrow O_P$: $\forall e_{oR} \in O_R$, $\exists e_{oP} \in O_P$, such that $C(e_{oR}) \equiv C(e_{oP})$; "≡" means the LHS concept is equivalent to the RHS concept. Clearly, this implies that $|O_R| = |O_P|$, i.e., the number of output parameters required by the desired Web service requested by the server requester is equal to the number of output parameters required by the Web service provided by the service publisher;

$f_{output\text{-}L1}$: a 1-1 mapping from $O_R \rightarrow O_P$: $\forall e_{oR} \in O_R$, $\exists e_{oP} \in O_P$, such that either one of these relation holds, $C(e_{oR}) \equiv C(e_{oP})$ or, $C(e_{oR}) < C(e_{oP})$; and there exists at least one pair of $e_{oR} \in O_R$, $e_{oP} \in O_P$ such that the < relation holds. "<" means the LHS concept is a sub-concept or sub-class of the RHS concept. Clearly, this implies that $|O_R| = |O_P|$;

$f_{output\text{-}L2}$: a 1-1 mapping from $O_R \rightarrow O_P$: $\forall e_{oR} \in O_R$, $\exists e_{oP} \in O_P$, such that either one of these relation holds, $C(e_{oR}) \equiv C(e_{oP})$ or, $C(e_{oR}) < C(e_{oP})$ or, $C(e_{oR}) > C(e_{oP})$; and there exist at least one pair of $e_{oR} \in O_R$,, $e_{oP} \in O_P$ such that the > relation holds.

"$>$" means the LHS concept is a super concept or super class of the RHS concept. Clearly, this implies that $|O_R| = |O_P|$.

Again, $f_{output\text{-}exact}$, $f_{output\text{-}L1}$ and $f_{output\text{-}L2}$ have similar implications as the three input-testing functions. Clearly, an "output-exact" matching is more preferable than an "output-L1" matching, which is more desirable than an "output-L2" matching.

Based on the above notations, the matching algorithm can then be described in Figure 3:

Algorithm 4.1

Input: Web service request description file (.owl)
current service descriptions from the SWS repository

Output: a string value from the set
{"exact","level-1","level-2","failed"}

Method: build I_R, O_R using Web service request description file;
build I_P, O_P using the current SWS repository;
if $|I_R| \neq |I_P|$ or $|O_R| \neq |O_P|$ return "failed";
else if ($\exists f_{input\text{-}exact}$ and $\exists f_{output\text{-}exact}$) return "exact";
else if ($\exists f_{input\text{-}exact}$ and $\exists f_{output\text{-}L1}$) return "level-1";
else if ($\exists f_{input\text{-}L1}$ and $\exists f_{output\text{-}exact}$) return "level-1";
else if ($\exists f_{input\text{-}L1}$ and $\exists f_{output\text{-}L1}$) return "level-1";
else if ($\exists f_{input\text{-}exact}$ and $\exists f_{output\text{-}L2}$) return "level-2";
else if ($\exists f_{input\text{-}L1}$ and $\exists f_{output\text{-}L2}$) return "level-2";
else if ($\exists f_{input\text{-}L2}$ and $\exists f_{output\text{-}L2}$) return "level-2";
else if ($\exists f_{input\text{-}L2}$ and $\exists f_{output\text{-}L1}$) return "level-2";
else if ($\exists f_{input\text{-}L2}$ and $\exists f_{output\text{-}exact}$) return "level-2";
else return "failed";

Fig. 3. Matchmaking Algorithm based on inputs/outputs

The algorithm starts by reading the service request file that is submitted by the service requester. This file describes the desired Web service using OWL-S (for detail example see Chapter 5), and its input and output parameters are concepts/classes defined in the shared domain specific ontology. After reading this requirement description, the matchmaking engine starts to scan the current SWS repository, the first Web service description is extracted from this directory, the detailed input and output concepts that are

supported by this Web service are also read in by the matchmaking engine. At this stage, all the necessary information is ready for the comparison.

Next, the engine tests how many input parameters that the service requester would like to provide and how many input parameters are in fact required by the current candidate, if this number does not match, the matchmaking engine skips this candidate and moves on to the next service description recorded in the SWS repository. Otherwise, the engine moves on to test the number of output parameters by checking whether the number of the output parameters required by the service requester is different from the number of the output parameters given by the current candidate. Again, if this number does not match, the engine skips this candidate and moves on to the next one.

If the number of input parameters and output parameters are matching, the engine continues to check the degree of matching by testing "`exact`" matching first. If it does not exist, "`Level-1`" matching is then tested, and "`Level-2`" matching is finally tested if "`Level-1`" testing fails.

The current Web service will be added to the candidate set if at least a "`Level-2`" matching exists in both the input set and the output set. In the worst case, i.e., "`Level-2`" matching cannot be confirmed in either input or output set, the algorithm will declare a "`failed`" match and will not include the current Web service into the candidate set. In this case, the engine will move on to the next Web service description in the current SWS repository. Intuitively, a "`failed`" match very likely means that at least one concept from either the input parameter set or the output parameter set (provided by the service requester) does not have any relationship with the any of the concepts used in the input parameter set and output parameter set of the current Web service description. Once every Web service description in the SWS repository is tested, the matchmaking engine stops and returns the resulting candidate set.

4.3 Discussion and Comparison to other Algorithms

Clearly, the above proposed matching algorithm considers mainly the matching between inputs and outputs. Other factors can also be taken into account. For example, in [23], the service category is also considered as a matching criterion. In the proposed architecture, it is not hard to make this extension. Also, one might suggest consider preconditions and effects of the Web service, however at this stage, at pointed out in [23], preconditions and effects are not yet sufficiently standardized for being considered by a matching algorithm.

"`Level-1`" and "`Level-2`" matching results in the proposed algorithm are mainly based on the results of subsumption relationship of the concepts. Generally, concept `A` is subsumed by concept `B` if concept `B` denotes a more general concept than `A`. Therefore, if input `A` in the request profile is subsumed by input `B` in the candidate service profile, it is then clear that input `A` can provide more specific information then that is required. It is therefore safe to assume that the service can still be executed properly. This situation is captured in the "`Level-1`" matching situation.

It is also clear that the direction of this subsumption relationship is important: in the case where input `A` in the request profile subsumes input `B` in the candidate service profile, it might happen that the advertised service requires some specific details for the proper execution which input `A` cannot provide. For a situation like this, our matching algorithm still includes this service as a potential candidate and let the service requester to decide whether this service is usable or not. To distinguish this case, we call it "`Level-2`" matching, therefore, a `Level-2` matching may not be as appropriate as a `Level-1` matching. Form this perspective, this matching algorithm can be seen as an extension to the algorithm that is proposed in [20]. For example, the matching algorithm presented in [20] concludes a partial match as long as there exists an inheritance relationship between the requester's concepts and the ones offered by the service provider (given that these concepts do not equivalent to each other). However, it fails to consider the direction of this inheritance relationship. As discussed above, this direction directly results in the difference between the `Level-1` and `Level-2` matches.

As a summary, the output from our matching algorithm is a set of potential candidates. This set includes `exact` matching results, `Level-1` as well as `Level-2` matching results. It is now up to the requester to decide the QoS of these candidates. A system for measuring the QoS can certainly be implemented, as suggested in [21]. This is one extensible part of the proposed architecture.

CHAPTER 5

IMPLEMETATION: A BIOINFORMATICS EXAMPLE

This chapter presents an implementation of the architecture described in Chapter 3. Since a domain specific ontology is always needed, a small ontology in the area of bioinformatics is created. The related development tools are discussed first and each component in the discovery architecture is then implemented with examples presented. Also, since the idea of semantic Web services is relatively new, several semantic Web services based on the example bioinformatics ontology are created for the purpose of prototyping the proposed architecture and simulating the execution of the suggested matchmaking algorithm. However, the whole architecture and matchmaking algorithm can certainly function for real life semantic Web services described using practical domain specific ontologies.

5.1 Related Development Tools

As discussed in the previous chapters, current research trend to solve the challenge of Web service discovery is to add semantic information to the description of a given Web service. As far as the implementation is concerned, there have been a number of efforts which are quite successful, noticeably RDF, RDF Schema (RDFS), DAML, OWL and OWL-S. Keeping the concepts straight between RDF, RDFS, DAML/DAML-OIL, OWL and OWL-S can be difficult, therefore in this section, an overview of these development tools and languages are presented before the implementation work is discussed in more detail.

The most relevant and most fundamental tool is the Resource Description Framework (RDF) [29] which is proposed by W3C as a standard for exchanging metadata, and a key technology for the W3C's Semantic Web initiative. The basic RDF data model consists of three object types: resource, property and statements. Resources are the central concept of RDF. They are used to describe anything, from Web pages to human being. Properties express specific aspects, characteristics, attributes, or relations used to describe a resource. Statements are composed of a specific resource together with a named property and the value of that property for that resource. The value can be another resource or a literal, which is a primitive term that is not evaluated by an RDF processor. A RDF model is a set of RDF statements. An example of a statement is given in Figure 4.

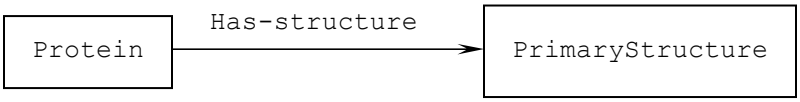

Fig. 4. An RDF statement

RDF was the first language specified by the W3C for representing semantic information about arbitrary resources. The primary purpose of RDF is to structure and describe existing data and it is typically enriched by an RDF Schema (RDFS) [30]. RDFS is a W3C candidate recommendation for an extension to RDF to describe RDF vocabularies. RDFS includes classes and properties, as well as range and domain constraints on properties. It provides inheritance hierarchies for both classes and properties. RDFS can be used to create ontologies, but it is purposefully light-weighted with less expressive power.

Upon the release of RDFS, users began requesting additional features, including data types, enumerations and the ability to define properties more rigorously. Meanwhile, other efforts in the research community were already examining exactly these sorts of features. For instance, DAML (DARPA Agent

Markup Language) [31] is an effort to add more features to RDF and therefore make it more expressive. Others included the following:

- MCF - Meta Content Framework.
- Ontobroker
- On-To-Knowledge
- OIL - Ontology Inference Layer
- SHOE - Simple HTML Ontology Extensions
- XOL

Instead of continuing with separate ontology languages for the Semantic Web, a group of researchers, including many of the main participants in both the OIL and DAML-ONT efforts, got together in the Joint US/EU ad hoc Agent Markup Language Committee to create a new Web ontology language. This language, DAML+OIL (DARPA Agent Markup Language in conjunction with the Ontology Inference Layer [31]), is built on both OIL and DAML-ONT, was submitted to the W3C as a proposed basis for OWL (Web Ontology Language) [32], and was subsequently selected as the starting point for OWL.

Based on RDF and RDF Schema, OWL – the successor DAML+OIL – increases the level of expressiveness with a richer vocabulary but retaining the decidability. However, it is primarily used to describe content. To describe the semantics of services in order to facilitate their discovery, an upper ontology for the description of Web services named DAML Services (DAML-S) had been introduced [9]. Using DAML-S, the capacity of the service can be completely described.

Recently DAML-S has been updated and renamed to OWL-Service (OWL-S) [11], which attracts much of the attention and becomes the standard of describing the capacity of a given service [33]. More specifically, OWL-S is an OWL-based Web service ontology, it supplies Web Service providers with a core set of markup language constructs for describing the properties and capabilities of their Web

Services in unambiguous, computer-interpretable form. OWL-S leverages on OWL to accomplish the following:

- support capability based discovery of Web services;
- Support automatic composition of Web Services;
- Support automatic invocation of Web services.

In this research, for the development of the prototype, several fictitious web service publishers are assumed to agree on the small bioinformatics ontology, and each of them will provide some Web services in the area of bioinformatics. OWL-S Upper `ServiceProfile` ontology [34] and concepts from the small bioinformatics ontology (written in OWL) are used to "markup" these Web services, i.e., to add semantics to these Web services. These details will be discussed in the next several sections.

5.2 A Simple Bioinformatics Ontology

One of the core components of semantic Web service infrastructure is a domain specific ontology. The terms/concepts that are expressed in the advertisements and requests have to be defined in a domain specific ontology. More specifically, an ontology is the attempt to formulate an exhaustive and rigorous conceptual schema within a given domain, typically a hierarchical data structure containing all the relevant entities and their relationship and rules within that domain. As the first step of the prototyping development, a small ontology using the concepts from the area of bioinformatics is created. This section presents details of this example ontology.

In recent years, bioinformatics emerged as a matured community in which computational techniques are frequently applied to vast amount of data and knowledge resources in order to answer complex biological questions. These data and resources are widely distributed, highly heterogeneous and highly diverse; all these characteristics make this area an ideal candidate for Web based tools. Examples along this direction can be found in [35-37].

As discussed in Section 5.1, RDFS can be used to create ontology, but OWL has become the standard for developing ontologies. For this reason, the small ontology in this prototyping development is created using OWL. It involves the basic and important concepts such as polymer, amino acids, polypeptide, protein, primary/secondary/tertiary structure, RNA, DNA, protein functions etc. Figure 5 is a segment of this ontology and the whole ontology written in OWL is given in Appendix A.

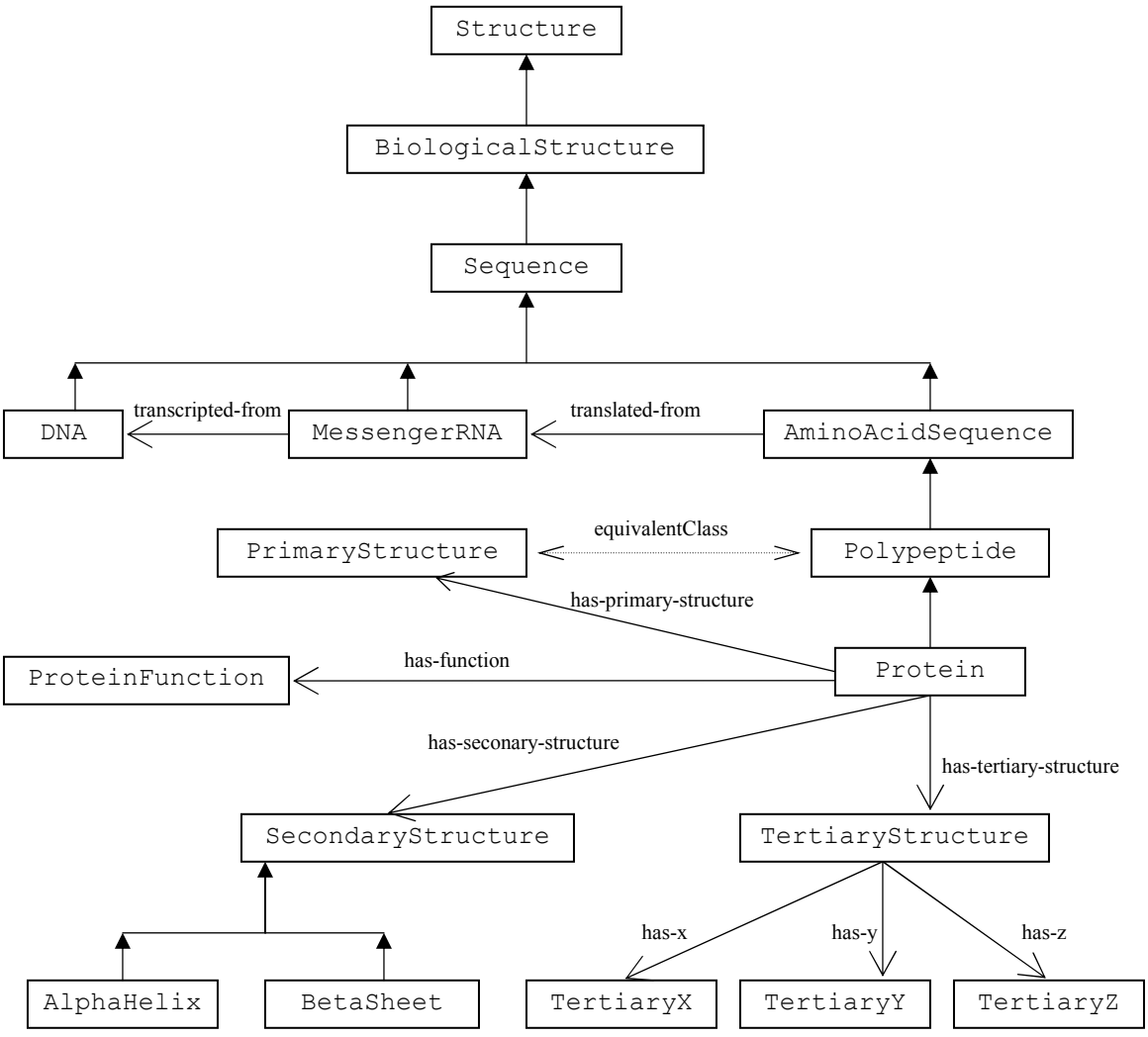

Fig. 5. A fragment of the example bioinformatics ontology

5.3 Semantic Web Services Based on the Example Ontology

To develop the prototype, the next step is to create several hypothetical Web services. Semantic information based on the example ontology has to be added to the service descriptions. To implement this, OWL-S language is used. The details are described in this section.

5.3.1 Hypothetical Web Services

Once the domain specific ontology is created, hypothetical Web services can be advertised using the terms that are defined in the ontology. Table 5 is a list of the hypothetical Web services that are proposed for the prototype development.

Table 5. A list of hypothetical Web services based on the example ontology

Web service name	Description and input/output terms

`getSubSequence()`	`[Description] get sub-sequence`
	`[Input]`
	`Sequence,SequenceLocation,SequenceLocation`
	`[Output] Sequence`

`getSeqLength()`	`[Desc] get sequence length`
	`[Input] Sequence`
	`[Output] SequenceLength`

`getSeqSimilarity()`	`[Description] get sequence similarity`
	`[Input] Sequence, Sequence`
	`[Output] SequenceSimilarity`

`getGlobalAlignment()`	`[Description] calculate global alignment`
	`[Input] Sequence, Sequence`
	`[Output] GlobalAlignment`

`getLocalAlignment()`	`[Description] calculate local alignment` `[Input] Sequence, Sequence`

	[Output] LocalAlignment
getProteinType()	[Description] get protein type [Input] DNA, AminoAcidSequence [Output] Protein
getProteinType()	[Description] get protein type [Input] MessengerRNA, AminoAcidSequence [Output] Protein
getProteinFunction()	[Description] get protein function [Input] PrimaryStructure [Output] ProteinFunction
getProteinFunction()	[Description] get protein function [Input] SecondaryStructure [Output] ProteinFunction
getProteinFunction()	[Description] get protein function [Input] TertiaryStructure [Output] ProteinFunction
getMainStructure()	[Description] get protein main structure [Input] Protein [Output] AminoAcidSequence
getSecondaryStructure()	[Description] get protein secondary structure [Input] AminoAcidSequence [Output] SecondaryStructure
getSecondaryStructure()	[Description] get protein secondary structure [Input] AminoAcidSequence,SequenceLocation [Output] SecondaryStructure

The next task is to add semantic information to the descriptions of these hypothetical services. A brief introduction to OWL-S upper service ontology is presented first and example of adding semantics to the services is discussed in Section 5.3.3.

5.3.2 OWL-S Upper Service Ontology

OWL-S is an OWL-based ontology for Web services (it is also called OWL-S upper ontology for Web services). It provides a markup language that can be used to describe Web services' properties and

capabilities in an unambiguous way that machine can interpret them. The aim of OWL-S is to provide means for developing semantic Web in which services can be discovered and also executed automatically without human interaction. Therefore, it satisfies the needs of this research perfectly.

OWL-S contains three major parts presenting the semantics of service: `ServiceProfile`, `ServiceModel` and `ServiceGrounding`. In these parts are described, what service does, how service works and how service is used. Among these three parts, the `ServiceProfile` is of particular interest to the development of the prototype in this research. It is a concise description of service, which is sent to the registry or service user like the broker. According to OWL-S Collation [11], the matchmaking done by the broker is based on the `ServiceProfile`, which gives the type of information needed to determine whether the service meets the needs of the service requester. Figure 6 shows the basic idea of this process.

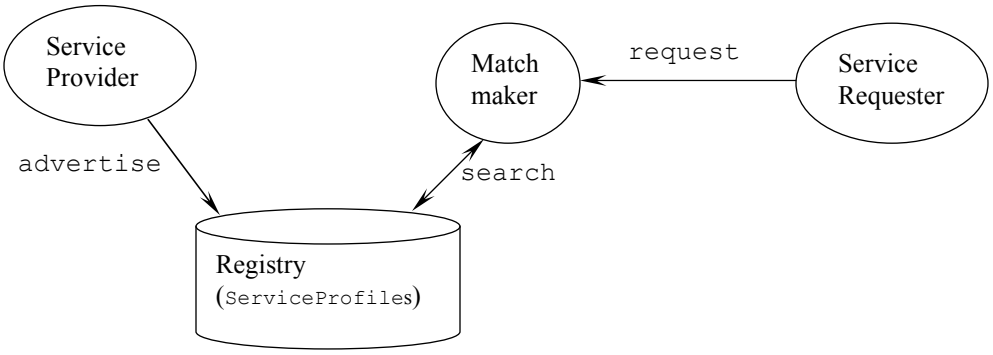

Fig. 6. `ServiceProfile` and service discovery

As the focus of this research is on the discovery and selection of Web services, let us concentrate on the upper OWL-S `ServiceProfile` ontology.

5.3.3 Using OWL-S `ServiceProfile` to Markup Service Descriptions

The `ServiceProfile` is divided into three main sections: (1) a textual description and contact information, which is mainly intended for human users, (2) a functional description of the service. This functional description describes the input and output of a service. Additionally, two sets of conditions are defined, namely preconditions, which have to hold before the service can be executed properly, and effects, which are conditions that hold after the successful execution of the service, i.e., postconditions. These four functional descriptions are also referred to as IOPE (Input, Output, Precondition, and Effects). The third type (3) is a set of additional properties that are used to describe the features of the service. For instance, service category, which is used to classify the service with respect to some ontology or taxonomy of services.

For a matchmaking engine, the most important part is the functional description part. More specifically, the OWL-S specification defines the semantic elements for advertising the functional description of a service with an instance of the class `Profile`. In the class `Profile`, RDF properties point to the IOPE elements, `Input`, `Output`, `Precondition` and `Effect`. Each of the four classes is a subclass of the class `Parameter`.

Figure 7 shows one example of using OWL-S `ServiceProfile` to create a profile of the "`getGlobalAlignment()`" Web service (see Table 5). The format used in Figure 7 is also used by "OWL-S/UDDI Matchmaker Web Interface" project developed by The Intelligent Software Agents Lab in Carnegie Mellon University [38]. This format captures the main functional characteristics of the service and it is also simple to implement and understand.

```
<?xml version='1.0' encoding='ISO-8859-1'?>

<!DOCTYPE uridef
[
   <!ENTITY rdf       "http://www.w3.org/1999/02/22-rdf-syntax-ns">
   <!ENTITY rdfs      "http://www.w3.org/2000/01/rdf-schema">
   <!ENTITY owl       "http://www.w3.org/2002/07/owl">
   <!ENTITY service   "http://www.daml.org/services/owl-s/1.0/Service.owl">
   <!ENTITY process   "http://www.daml.org/services/owl-s/1.0/Process.owl">
```

```
   <!ENTITY profile   "http://www.daml.org/services/owl-s/1.0/Profile.owl">
   <!ENTITY actor     "http://www.daml.org/services/owl-/1.0/ActorDefault.owl">
   <!ENTITY domainOnt
            "http://tinman.cs.gsu.edu/~lyu2/thesis/SimpleBioOntology.owl">
   <!ENTITY DEFAULT
            http://tinman.cs.gsu.edu/~lyu2/thesis/BioInformatics_ws_2_1.owl">
]>

<rdf:RDF
   xmlns:rdf=       "&rdf;#"
   xmlns:rdfs=      "&rdfs;#"
   xmlns:owl=       "&owl;#"
   xmlns:service=   "&service;#"
   xmlns:process=   "&process;#"
   xmlns:profile=   "&profile;#"
   xmlns:actor=     "&actor;#"
   xmlns:domainOnt= "&domainOnt;#"
   xmlns=           "&DEFAULT;#">

   <owl:Ontology about="">
      <owl:imports rdf:resource="&rdf;" />
      <owl:imports rdf:resource="&rdfs;" />
      <owl:imports rdf:resource="&owl;" />
      <owl:imports rdf:resource="&service;" />
      <owl:imports rdf:resource="&profile;" />
      <owl:imports rdf:resource="&process;" />
      <owl:imports rdf:resource="&actor;" />
      <owl:imports rdf:resource="&domainOnt;" />
   </owl:Ontology>

   <profile:Profile rdf:ID="BioInformatics_ws_4">

      <profile:serviceName>BioInformatics_ws_4</profile:serviceName>
      <profile:textDescription>some description</profile:textDescription>

      <profile:contactInformation>
         <actor:Actor rdf:ID="LiyangYu_WS4">
               <actor:name>liyang yu</actor:name>
               <actor:title>Research Programmer</actor:title>
               <actor:phone>404.395.1680</actor:phone>
               <actor:fax>na</actor:fax>
               <actor:email>lyu2@student.gsu.edu</actor:email>
               <actor:physicalAddress>cs,GSU</actor:physicalAddress>
                <actor:webURL>http://tinman.cs.gsu.edu/~lyu2/thesis
                </actor:webURL>
         </actor:Actor>
      </profile:contactInformation>

      <!-- Descriptions of the parameters that will be used by IOPEs -->
      <profile:hasInput rdf:resource="#input1-sequence"/>
      <profile:hasInput rdf:resource="#input2-sequence"/>
      <profile:hasOutput rdf:resource="#output1-globalAlignment"/>

   </profile:Profile>

   <process:Input rdf:ID="input1-sequence">
      <process:parameterType rdf:resource="&domainOnt;#Sequence"/>
   </process:Input>

   <process:Input rdf:ID="input2-sequence">
      <process:parameterType rdf:resource="&domainOnt;#Sequence"/>
   </process:Input>
```

```
   <process:UnConditionalOutput rdf:ID="output1-globalAlignment">
      <process:parameterType rdf:resource="&domainOnt;#GlobalAlignment"/>
   </process:UnConditionalOutput>

</rdf:RDF>
```

Fig. 7. Using OWL-S to describe `getGlobalAlignment` Web service

Other Web services listed in Table 5 have similar description profiles. Clearly, the description presented in Figure 7 is relatively simple; however it is enough for the current version of the matchmaking engine to work. More specifically, the inputs and outputs, together with the specific domain ontology are included in the profile. For example, resource "`input1-sequence`" is mapping to the term "`&domainOnt;#Sequence`" defined in the example ontology. For comparison, Figure 8 shows an OWL-S service profile (`bravoCarRental.owl`) taken from the "OWL-S/UDDI Matchmaker Web Interface" project:

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<!DOCTYPE uridef
[
  <!ENTITY rdf      "http://www.w3.org/1999/02/22-rdf-syntax-ns">
  <!ENTITY rdfs     "http://www.w3.org/2000/01/rdf-schema">
  <!ENTITY owl      "http://www.w3.org/2002/07/owl">
  <!ENTITY xsd      "http://www.w3.org/2001/XMLSchema">
  <!ENTITY service  "http://www.daml.org/services/owl-s/1.0/Service.owl">
  <!ENTITY profile  "http://www.daml.org/services/owl-s/1.0/Profile.owl">
  <!ENTITY actor    "http://www.daml.org/services/owl-s/1.0/ActorDefault.owl">
  <!ENTITY process  "http://www.daml.org/services/owl-s/1.0/Process.owl">
  <!ENTITY country  "http://www.daml.org/services/owl-s/1.0/Country.owl">
  <!ENTITY concepts "http://www.daml.org/services/owl-s/1.0/Concepts.owl">
  <!ENTITY addParam
      "http://www.daml.org/services/owl-s/1.0/ProfileAdditionalParameters.owl">
  <!ENTITY profileHierarchy
      "http://www.daml.org/services/owl-s/1.0/ProfileHierarchy.owl">
  <!ENTITY bc_service
      "http://www.daml.org/services/owl-s/1.0/BravoCarService.owl">
  <!ENTITY bc_process
      "http://www.daml.org/services/owl-s/1.0/BravoCarProcess.owl">
  <!ENTITY DEFAULT
      "http://www.daml.org/services/owl-s/1.0/BravoCarProfile.owl">
]>

<!-- This document uses entity types as a shorthand for URIs.
Download the source for a version with unexpanded entities. -->

<rdf:RDF
```

```
        xmlns:rdf=      "&rdf;#"
        xmlns:rdfs=     "&rdfs;#"
        xmlns:owl=      "&owl;#"
        xmlns:xsd=      "&xsd;#"
        xmlns:service=  "&service;#"
        xmlns:process=  "&process;#"
        xmlns:profile=  "&profile;#"
        xmlns:actor=    "&actor;#"
        xmlns:addParam= "&addParam;#"
        xmlns:profileHierarchy= "&profileHierarchy;#"
        xml:base="&DEFAULT;#"
        xmlns="&DEFAULT;#">

      <owl:Ontology rdf:about="">
          <owl:versionInfo>
            $Id: BravoAirProfile.owl,v 1.18 2003/12/11 01:57:41 martin Exp $
          </owl:versionInfo>
          <rdfs:comment>
            DAML-S Coalition: BravoAir Example for OWL-S Profile description
          </rdfs:comment>
          <owl:imports rdf:resource="&service;" />
          <owl:imports rdf:resource="&profile;" />
          <owl:imports rdf:resource="&process;" />
          <owl:imports rdf:resource="&actor;" />
          <owl:imports
rdf:resource="http://pericles.cimds.ri.cmu.edu:8080/owl/ProfileAdditionalParameters.owl"/>
          <owl:imports rdf:resource="&country;" />
          <owl:imports
rdf:resource="http://pericles.cimds.ri.cmu.edu:8080/owl/ProfileHierarchy.owl"/>
          <owl:imports
rdf:resource="http://pericles.cimds.ri.cmu.edu:8080/owl/matchmaker-test.owl"/>
        </owl:Ontology>

        <profile:Profile rdf:ID="Profile_BravoCar_ReservationAgent">
          <service:presentedBy
             rdf:resource="&bc_service;#BravoCar_ReservationAgent"/>
          <profile:has_process rdf:resource="&bc_process;#BravoCar_Process"/>

          <profile:serviceName>Bravo_Car_Rental</profile:serviceName>
          <profile:textDescription>
             This  service provide Car rental service.
          </profile:textDescription>

          <profile:contactInformation>
            <actor:Actor rdf:ID="BravoCar-reservation">
              <actor:name>Bravocar  Reservation department</actor:name>
              <actor:title>Reservation Representative</actor:title>
              <actor:phone>412 268 8780 </actor:phone>
              <actor:fax>412 268 5569 </actor:fax>
              <actor:email>Bravo@Bravoair.com</actor:email>
              <actor:physicalAddress>
                   Airstrip 2, Teetering Cliff Hights,
                   Florida 12321, USA
              </actor:physicalAddress>
              <actor:webURL>
                  http://www.daml.org/services/daml-s/2001/05/BravoAir.html
               </actor:webURL>
            </actor:Actor>
          </profile:contactInformation>

          <profile:contactInformation>
             … another contact information is omitted here…
          </profile:contactInformation>
```

```
    <profile:serviceParameter>
      <addParam:GeographicRadius rdf:ID="BravoCar-geographicRadius">
       <profile:serviceParameterName>
         BravoAir Geographic Radius
       </profile:serviceParameterName>
       <profile:sParameter rdf:resource="&country;#UnitedStates"/>
      </addParam:GeographicRadius>
    </profile:serviceParameter>


    <profile:qualityRating>
      <profile:QualityRating rdf:ID="BravoCar-goodRating">
       <profile:ratingName>SomeRating</profile:ratingName>
       <profile:rating rdf:resource="&concepts;#GoodRating"/>
      </profile:QualityRating>
    </profile:qualityRating>

    <profile:serviceCategory>
      <addParam:NAICS rdf:ID="NAICS-category">
       <profile:value>Car Rental Service</profile:value>
       <profile:code>561599<profile:code>
      </addParam:NAICS>
    </profile:serviceCategory>

    <!-- Specification of the service category using UN-SPSC -->
    <profile:serviceCategory>
      <addParam:UNSPSC rdf:ID="UNSPSC-category">
       <profile:value>Car Rentals</profile:value>
       <profile:code>90121500</profile:code>
      </addParam:UNSPSC>
    </profile:serviceCategory>

    <!-- Descriptions of IOPEs -->
    <profile:hasInput rdf:resource="#RentalLocation"/>
    <profile:hasInput rdf:resource="#CarDescription"/>
    <profile:hasInput rdf:resource="#StartDate"/>
    <profile:hasInput rdf:resource="#Days"/>
    <profile:hasOutput rdf:resource="#RentalAgreement"/>

  </profile:Profile>

  <process:Input rdf:ID="RentalLocation">
    <process:parameterType rdf:resource="&concepts;#Airport"/>
  </process:Input>

  <process:Input rdf:ID="CarDescription">
    <process:parameterType rdf:resource="&concepts;#CarDescription"/>
  </process:Input>

  <process:Input rdf:ID="StartDate">
    <process:parameterType rdf:resource="&concepts;#RentalDate"/>
  </process:Input>

  <process:Input rdf:ID="Days">
    <process:parameterType rdf:resource="&xsd;#integer"/>
  </process:Input>

  <process:UnConditionalOutput rdf:ID="RentalAgreement">
    <process:parameterType rdf:resource="&concepts;#CarRentalAgreement"/>
  </process:UnConditionalOutput>

</rdf:RDF>
```

Fig. 8. `bravoCarRental.owl` from OWL-S/UDDI matchmaker Web interface project [38]

Besides the inputs and outputs, the profile in Figure 8 also includes the `qualityRating` and `serviceCategory` (including both the NAICS and UNSPSC taxonomies) information, for the purpose of the matchmaking algorithm that is presented in [23]. To extend the algorithm proposed in this research, similar (or even more) information can be added in the service profile to make finer and more accurate matchmaking results.

Now assume that some fictitious Web service providers decide to advertise the services listed in Table 5 on their Web sites using the profile such as presented in Figure 7 (and ideally, some real service providers also published their real Web services using OWL profile language), it is then up to the Web crawler to find and collect the related information into the SWS repository. This is discussed in next section.

5.4 Using Web Crawler to Build the SWS Repository

The role of the Web crawler is to find the advertisements such as those presented in Section 5.3. It is different from the crawlers used in google.com and other search engines in that it only searches and collects the service profiles that are written in OWL-S. For this reason, a better name therefore could be OWL-S crawler. An example of a DAML crawler is given in [39], another crawler example, the SHOE (Simple Html Ontology Extension) crawler called Expose, is available in [40]. This section discussed the development of the OWL-S crawler in the prototype system and presents the searching results both from the published sites and UDDI registries.

5.4.1 Developing the Crawler: Java and Jena APIs

The OWL-S crawler is a robot that searches for Web pages with OWL-S markup, reads the relevant information from them and loads it into SWS repository. This multi-threaded OWL-S crawler is written in Java and it is initialized by given it a starting URL as the current page. When the robot reads a page, it decides whether this page is an owl page or not. If it is, the crawler extracts the necessary information and stores it in SWS repository. When the crawler finishes extracting information (or if this page is not an owl page), it then identifies all URLs within the current page and creates a new robot (thread) for each page and repeats the same process again. This process is described in Figure 9.

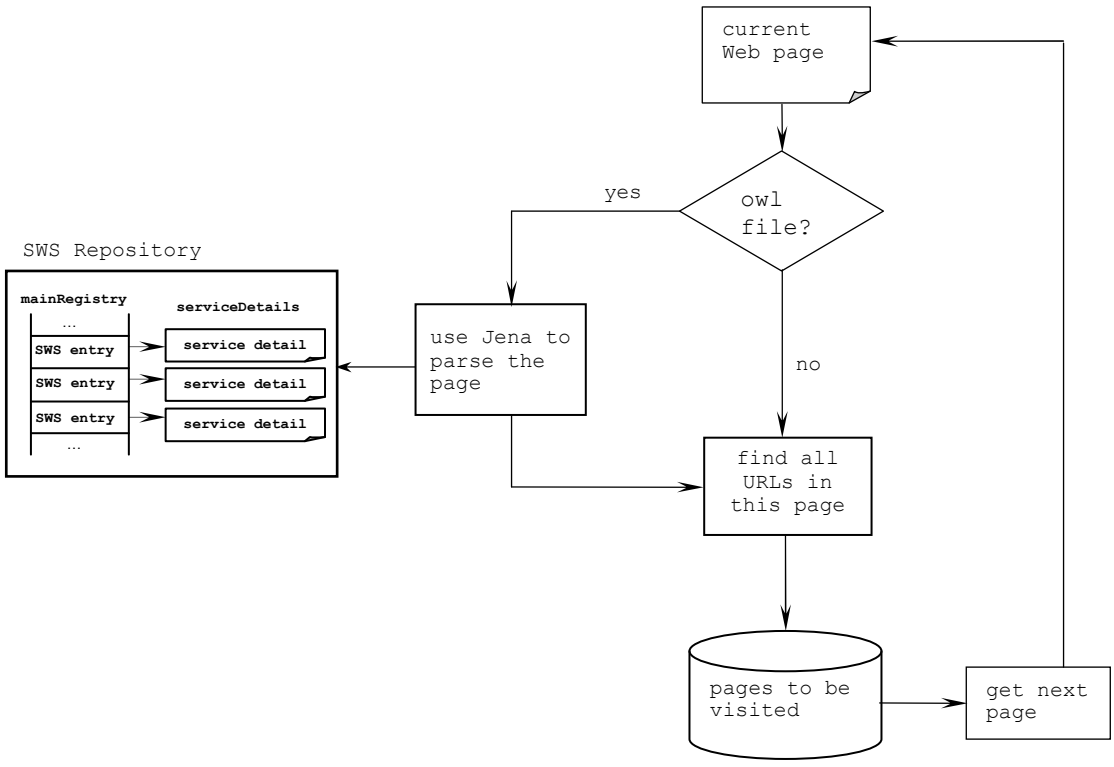

Fig. 9. basic flow of OWL-S crawler

There are several important issues that are needed to be discussed here. The first issue is the parsing of an owl page: once the crawler recognizes that the current page is indeed an owl page, it needs to read the OWL-S statements contained in this page to populate the SWS repository. More specifically, the input and output terms have to be extracted and mapped to the given domain ontology. Jean [41] is selected to implement the parsing of the owl profile: it is a Java framework for building semantic Web applications; it provides a programmatic environment for RDF, RDFS and OWL/OWL-S, including a rule-based inference engine. More details about Jean can be found in its official Web site at [41], and the parsing details can be found in the code that is developed for this research.

Another important issue is the multithreading issue. Currently, the crawler finds all the URLs that are contained in the current page, and for each one of these URLs, a new crawler is created as a new thread to explore the page represented by the given URL. For this page, the crawler will again parse the owl file (if this page is indeed an owl profile) and will extract all the URLs that are maintained by this page and start to generate new crawlers (threads) for these URLs. Therefore, the very initial crawler will in fact spawn many new crawlers to accomplish the final goal. An important problem of this implementation is that the system will quick run out of resources, given the vast amount of information on the Internet. In the current version, the number of threads that can exist simultaneously is limited by some configuration parameter to control and manage the use of the system resources. This method is relatively simple; however, many threads (new crawlers) will have to "sleep" to wait for their turns, meaning that visiting the Web (i.e., updating the SWS repository) can potential take quite long to finish.

5.4.2 Building SWS Repository by Visiting the Internet

This section presents the results by running the OWL-S on the Internet. As discussed earlier, given the limitation on the system resources and timing constraints, it is unrealistic to expect it to find "all" the service profiles that are published on the Web. In order to find as many profiles as possible,

some profile examples presented in the "OWL-S/UDDI Matchmaker Web Interface" projects [38] are moved to the Web site of this research (this Web site is used as the initial URL for the crawler, `http://tinman.cs.gsu.edu/~lyu2/thesis/research.html`) to guarantee the crawler will find them within a short period of time. Figure 10 shows a segment of the crawler log file; Table 6 and 7 show the resulting `mainRegistry` and the `serviceDetail` table in the SWS repository after the crawler finishes the work.

```
Semantic Web Services Indexation and Discovery: OWL-S crawler log file
DATE: Thu Jan 05 09:32:41 EST 2006

>>> starting URL:http://tinman.cs.gsu.edu/~lyu2/thesis/research.html
>>> max number of pages to visit:  5000
>>> max number of existing threads:  4

>>> connecting to ORACLE database (jdbc:oracle:thin:@tinman.cs.gsu.edu:1521:sid9ir2)
>>> Connection to ORACLE is successful!

>>> [thread-0:http://tinman.cs.gsu.edu/~lyu2/thesis/research.html] has 3 links.
[thread-0] visiting-> http://tinman.cs.gsu.edu/~lyu2/thesis/title.html
[thread-0] visiting-> http://tinman.cs.gsu.edu/~lyu2/thesis/research_leftPane.html
[thread-0] visiting-> http://tinman.cs.gsu.edu/~lyu2/thesis/research_introduction.html

>>> [thread-3:http://tinman.cs.gsu.edu/~lyu2/thesis/research_introduction.html] has 8 links.
[thread-3] visiting-> http://uddi.org
[thread-3] visiting-> http://www.w3.org/2002/ws/desc/
[thread-3] visiting-> http://www.w3.org/2000/xp/Group/
[thread-3] visiting-> http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-
84A9809EC588EF21

>>> [thread-2:http://tinman.cs.gsu.edu/~lyu2/thesis/research_leftPane.html] has 4 links.
[thread-2] visiting-> http://tinman.cs.gsu.edu/~lyu2/thesis/research_background.html
[thread-2] visiting-> http://tinman.cs.gsu.edu/~lyu2/thesis/research_plan.html
[thread-2] visiting-> http://tinman.cs.gsu.edu/~lyu2/thesis/research_implementation.html

>>> [thread-1:http://tinman.cs.gsu.edu/~lyu2/thesis/title.html] has 1 links.
[thread-1] visiting-> http://tinman.cs.gsu.edu/~raj

>>> [thread-4:http://uddi.org] has 1 links.
[thread-4] visiting-> registry.oasis-open.org

>>> [thread-8:http://tinman.cs.gsu.edu/~lyu2/thesis/research_background.html] has 2 links.
[thread-8] visiting-> http://infomesh.net/2001/swintro/

>>> [thread-7:http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21] has
32 links.
[thread-7] visiting-> American.com
[thread-7] visiting-> http://www.sciam.com/xml/sciam.xml
[thread-7] visiting-> http://oas-central.realmedia.com/RealMedia/ads/
```

Fig. 10. Segment of the crawler log

Table 6. mainRegistry table

notice: 1. ~ is defined as http://tinman.cs.gsu.edu/~lyu2/thesis
2. UDDIEntry field is omitted since all of them is null

serviceURL	ontologyURL	serviceName	contactEmail
~/BioInformatics_ws_1_1.owl	~/SimpleBioOntology.owl	BioInformatics_ws_1	…
~/BioInformatics_ws_1_3.owl	~/SimpleBioOntology.owl	BioInformatics_ws_3	…
~/BioInformatics_ws_1_2.owl	~/SimpleBioOntology.owl	BioInformatics_ws_2	…
~/BioInformatics_ws_2_1.owl	~/SimpleBioOntology.owl	BioInformatics_ws_4	…
~/BioInformatics_ws_2_2.owl	~/SimpleBioOntology.owl	BioInformatics_ws_5	…
~/BioInformatics_ws_3_1.owl	~/SimpleBioOntology.owl	BioInformatics_ws_6	…
~/BioInformatics_ws_3_2.owl	~/SimpleBioOntology.owl	BioInformatics_ws_7	…
~/BioInformatics_ws_3_3.owl	~/SimpleBioOntology.owl	BioInformatics_ws_8	…
~/BioInformatics_ws_3_4.owl	~/SimpleBioOntology.owl	BioInformatics_ws_9	…
~/BioInformatics_ws_3_5.owl	~/SimpleBioOntology.owl	BioInformatics_ws_10	…
~/BioInformatics_ws_4_1.owl	~/SimpleBioOntology.owl	BioInformatics_ws_11	…
~/BioInformatics_ws_4_2.owl	~/SimpleBioOntology.owl	BioInformatics_ws_12	…
~/BioInformatics_ws_4_3.owl	~/SimpleBioOntology.owl	BioInformatics_ws_13	…
~/BioInformatics_ws_2_3.owl	~/SimpleBioOntology.owl	BioInf_ws_alignment1	…
~/BioInformatics_ws_2_4.owl	~/SimpleBioOntology.owl	BioInf_ws_alignment2	…
~/congoStockBroker.owl	http://www.daml.org/services/owl-s/1.0/Concepts.owl	Bravo_Car_Rental	John_Doe @Bravoair.com
~/bravoAirline.owl	null	BravoAir_ ReservationAgent	Bravo @Bravoair.com
~/abcBookFinder.owl	http://www.daml.org/services/owl-s/1.0/Concepts.owl	ABC_Books	John_Doe @Bravoair.com
~/bravoCarRental.owl	http://www.daml.org/services/owl-s/1.0/Concepts.owl	Bravo_Car_Rental	John_Doe @Bravoair.com
http://www.daml.org/services/owl-s/1.0/BravoAirProcess.owl	http://www.daml.org/services/owl-s/1.0/Concepts.owl	null	null
~/dreamInsurance.owl	http://www.daml.org/services/owl-	Bravo_Car_Rental	John_Doe

	s/1.0/Concepts.owl		@Bravoair.com

Table 7. serviceDetail table

serviceURL	parameterType	parameterClass
~/BioInformatics_ws_1_1.owl	INPUT-0	SequenceLocation
~/BioInformatics_ws_1_1.owl	INPUT-1	SequenceLocation
~/BioInformatics_ws_1_1.owl	INPUT-2	Sequence
~/BioInformatics_ws_1_1.owl	OUTPUT-0	Sequence
~/BioInformatics_ws_1_2.owl	INPUT-0	Sequence
~/BioInformatics_ws_1_2.owl	OUTPUT-0	SequenceLength
~/BioInformatics_ws_1_3.owl	INPUT-0	Sequence
~/BioInformatics_ws_1_3.owl	INPUT-1	Sequence
~/BioInformatics_ws_1_3.owl	OUTPUT-0	SequenceSimilarity
~/BioInformatics_ws_2_1.owl	INPUT-0	Sequence
~/BioInformatics_ws_2_1.owl	INPUT-1	Sequence
~/BioInformatics_ws_2_1.owl	OUTPUT-0	GlobalAlignment
~/BioInformatics_ws_2_2.owl	INPUT-0	Sequence
~/BioInformatics_ws_2_2.owl	INPUT-1	Sequence
~/BioInformatics_ws_2_2.owl	OUTPUT-0	LocalAlignment
~/BioInformatics_ws_3_1.owl	INPUT-0	DNA
~/BioInformatics_ws_3_1.owl	INPUT-1	AminoAcidSequence
~/BioInformatics_ws_3_1.owl	OUTPUT-0	Protein
~/BioInformatics_ws_3_2.owl	INPUT-0	AminoAcidSequence
~/BioInformatics_ws_3_2.owl	INPUT-1	MessengerRNA
~/BioInformatics_ws_3_2.owl	OUTPUT-0	Protein
~/BioInformatics_ws_3_3.owl	INPUT-0	PrimaryStructure
~/BioInformatics_ws_3_3.owl	OUTPUT-0	ProteinFunction

~/BioInformatics_ws_3_4.owl	INPUT-0	SecondaryStructure
~/BioInformatics_ws_3_4.owl	OUTPUT-0	ProteinFunction
~/BioInformatics_ws_3_5.owl	INPUT-0	TertiaryStructure
~/BioInformatics_ws_3_5.owl	OUTPUT-0	ProteinFunction
~/BioInformatics_ws_4_1.owl	INPUT-0	Protein
~/BioInformatics_ws_4_1.owl	OUTPUT-0	AminoAcidSequence
~/BioInformatics_ws_4_2.owl	INPUT-0	AminoAcidSequence
~/BioInformatics_ws_4_2.owl	OUTPUT-0	SecondaryStructure
~/BioInformatics_ws_4_3.owl	INPUT-0	SequenceLocation
~/BioInformatics_ws_4_3.owl	INPUT-1	AminoAcidSequence
~/BioInformatics_ws_4_3.owl	OUTPUT-0	SecondaryStructure
~/BioInformatics_ws_2_3.owl	INPUT-0	Sequence
~/BioInformatics_ws_2_3.owl	INPUT-1	Sequence
~/BioInformatics_ws_2_3.owl	OUTPUT-0	Alignment
~/BioInformatics_ws_2_4.owl	INPUT-0	AminoAcidSequence
~/BioInformatics_ws_2_4.owl	INPUT-1	AminoAcidSequence
~/BioInformatics_ws_2_4.owl	OUTPUT-0	GlobalAlignment
~/congoStockBroker.owl	INPUT-0	CompanyTickerSymbol
~/congoStockBroker.owl	INPUT-1	CreditCard
~/congoStockBroker.owl	INPUT-2	Integer
~/congoStockBroker.owl	OUTPUT-0	Stocks
~/abcBookFinder.owl	INPUT-0	ISBN
~/abcBookFinder.owl	OUTPUT-0	String
~/bravoCarRental.owl	INPUT-0	Airport
~/bravoCarRental.owl	INPUT-1	CarDescription
~/bravoCarRental.owl	INPUT-2	Integer
~/bravoCarRental.owl	INPUT-3	RentalDate
~/bravoCarRental.owl	OUTPUT-0	CarRentalAgreement
~/dreamInsurance.owl	INPUT-0	VIN

~/dreamInsurance.owl	OUTPUT-0	CarInsurance
http://www.daml.org/services/owl-s/1.0/BravoAirProcess.owl	INPUT-0	Password
http://www.daml.org/services/owl-s/1.0/BravoAirProcess.owl	INPUT-1	Confirmation
http://www.daml.org/services/owl-s/1.0/BravoAirProcess.owl	INPUT-2	FlightDate
http://www.daml.org/services/owl-s/1.0/BravoAirProcess.owl	INPUT-3	RoundTrip
http://www.daml.org/services/owl-s/1.0/BravoAirProcess.owl	INPUT-4	FlightItinerary
http://www.daml.org/services/owl-s/1.0/BravoAirProcess.owl	INPUT-5	AcctName
http://www.daml.org/services/owl-s/1.0/BravoAirProcess.owl	INPUT-6	FlightDate
http://www.daml.org/services/owl-s/1.0/BravoAirProcess.owl	INPUT-7	ReservationNumber
http://www.daml.org/services/owl-s/1.0/BravoAirProcess.owl	INPUT-8	Airport
http://www.daml.org/services/owl-s/1.0/BravoAirProcess.owl	INPUT-9	Airport
http://www.daml.org/services/owl-s/1.0/BravoAirProcess.owl	OUTPUT-0	ReservationNumber
http://www.daml.org/services/owl-s/1.0/BravoAirProcess.owl	OUTPUT-1	FlightItineraryList
http://www.daml.org/services/owl-s/1.0/BravoAirProcess.owl	OUTPUT-2	AcctName
http://www.daml.org/services/owl-s/1.0/BravoAirProcess.owl	OUTPUT-3	FlightItinerary

5.4.3 Building the SWS Repository by Visiting UDDI Registries

Besides publishing the semantic Web service profiles on their Web sites, service providers can also elect to register their services in public/private UDDI registries and the relevant semantic information can be added to the UDDI entries to facilitate the discovery of these services. Therefore, the OWL-S crawler not only has to collects the profiles from published Web sites, but also has to search for semantically enhanced UDDI entries in public/private UDDI registries. This section presents the details of developing such functionalities so the OWL-S crawler can visit UDDI entries in searching for semantic Web service descriptions.

There have been a number of efforts that aim at adding semantic information to UDDI entries, noticeably the work presented in [10] and others are discussed in Section 2.1. However, these different methods are all research proposals – there have been no standards specifying how to add semantic information into UDDI registries. Also, none of these published research work shows enough technical details about how exactly the proposed method should be implemented; it is commonly true that the very same idea can be implemented in a number of different ways. This indeed makes it difficult to develop a robust crawler to collect the relevant information from UDDI entries.

Fortunately, these seemingly popular proposals (see Section 2.1 for details) share one common idea: the UDDI's `tModel` construct is used as the key enabler for adding semantic information. For instance, the work presented in [10] suggests the use of `input_tModel` and `output_tModel` to represent the input and output terms that are defined in some domain specific ontology. Therefore, without the help of any existing standard(s) about adding semantic information to UDDI registries, the UDDI registry crawler developed in this research uses the `input_tModel` and `output_tModel` as the key reference to search for semantically enhanced service descriptions in UDDI entries.

Another important fact is that the UDDI registry crawler is not a "real" crawler: it does not need to look for UDDI registries over the Internet, since all the public UDDI registries are known to the public.

Therefore, it is easier to implement the UDDI crawler than to implement a Web crawler described in previous section. Table 8 shows these registries.

Table 8. public UDDI registries

Registry name	URL
Microsoft's UDDI Business Registry Node	`uddi.microsoft.com`
IBM's UDDI Business Registry Node	`uddi.ibm.com`
SAP's UDDI Business Registry Node	`uddi.sap.com`
NTT Com's UDDI Business Registry Node	`http://www.ntt.com/uddi/index-e.html`
Microsoft's Test UDDI Registry	`test.uddi.microsoft.com`
IBM's Test UDDI Registry	`uddi.ibm.com/testregistry/registry.html`
SAP's Test UDDI Registry	`udditest.sap.com`

For the purpose of proof of concept, the current implementation of the UDDI crawler is to search only in IBM's test UDDI registry, using the `input_tModel` and `output_tModel` as the key references. In order to clearly illustrate the process of searching this UDDI registry, Figure 11 – 14 shows a step-by-step manual process of searching the IBM test UDDI registry (including the search result), and Figure 15 shows the result obtained by using the current UDDI crawler – the service key is correctly reported by the crawler. This "`CookingBook`" service is clearly some testing work; it is therefore not added to the SWS repository.

Fig. 11. IBM's test UDDI registry

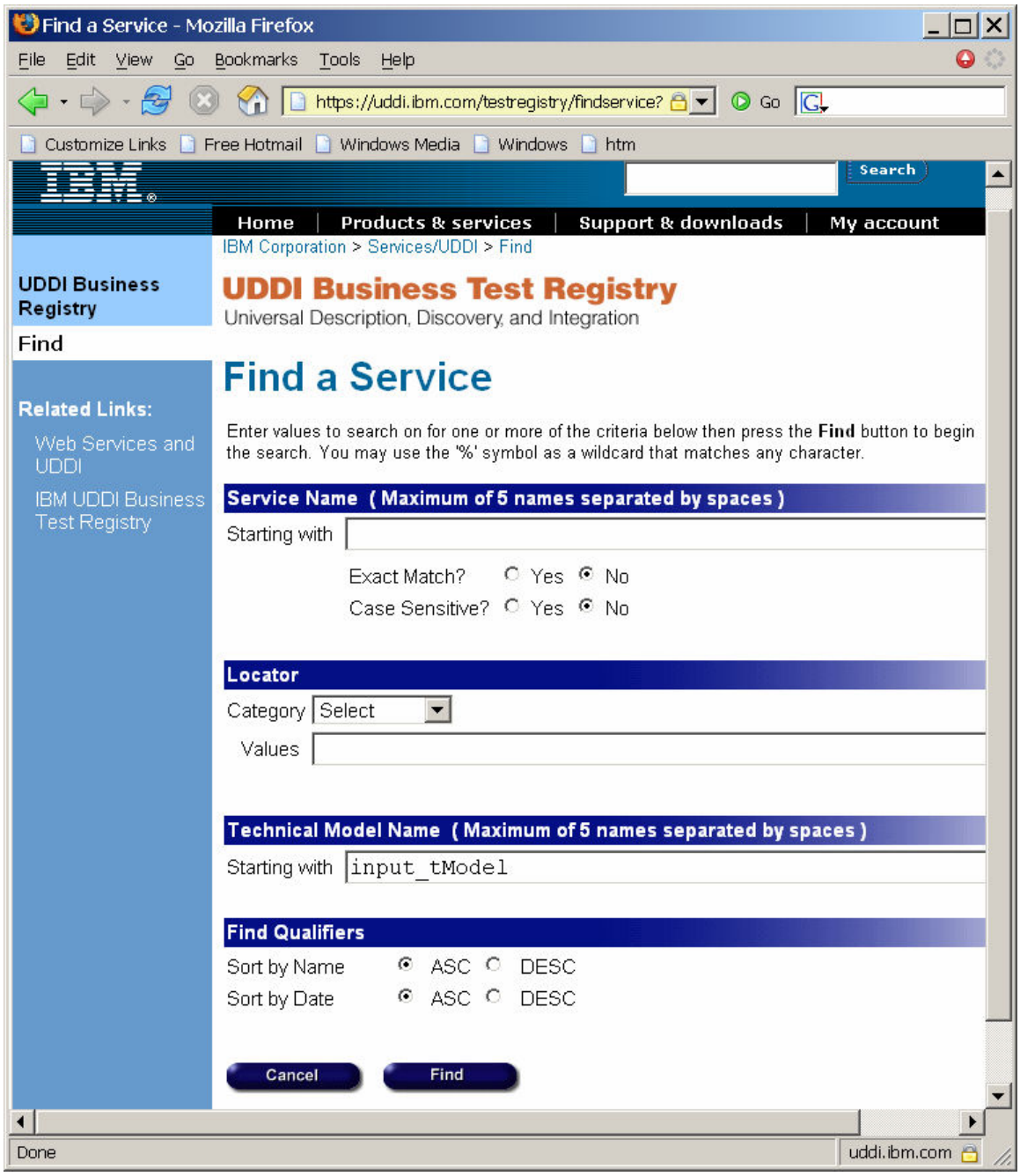

Fig. 12. search by using `input_tModel` as reference

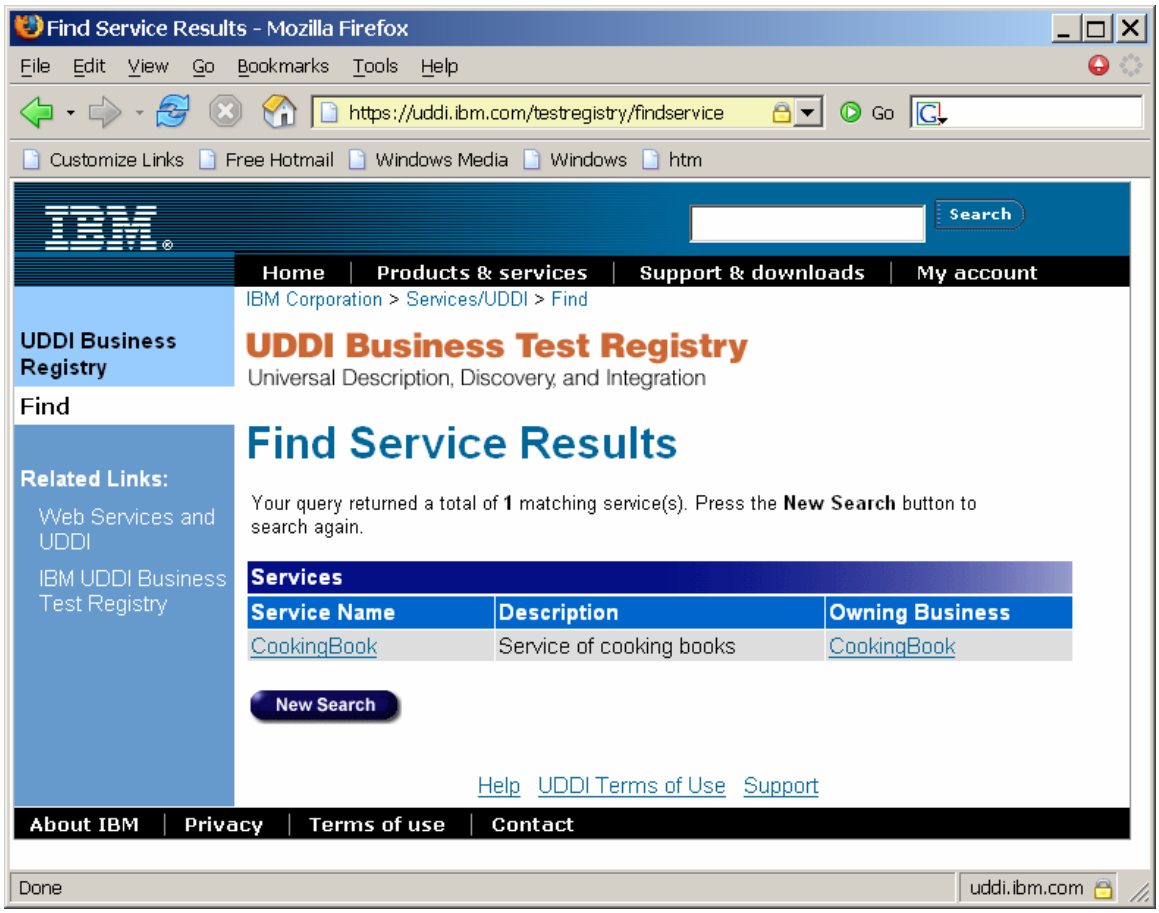

Fig. 13. a web service that has semantic information

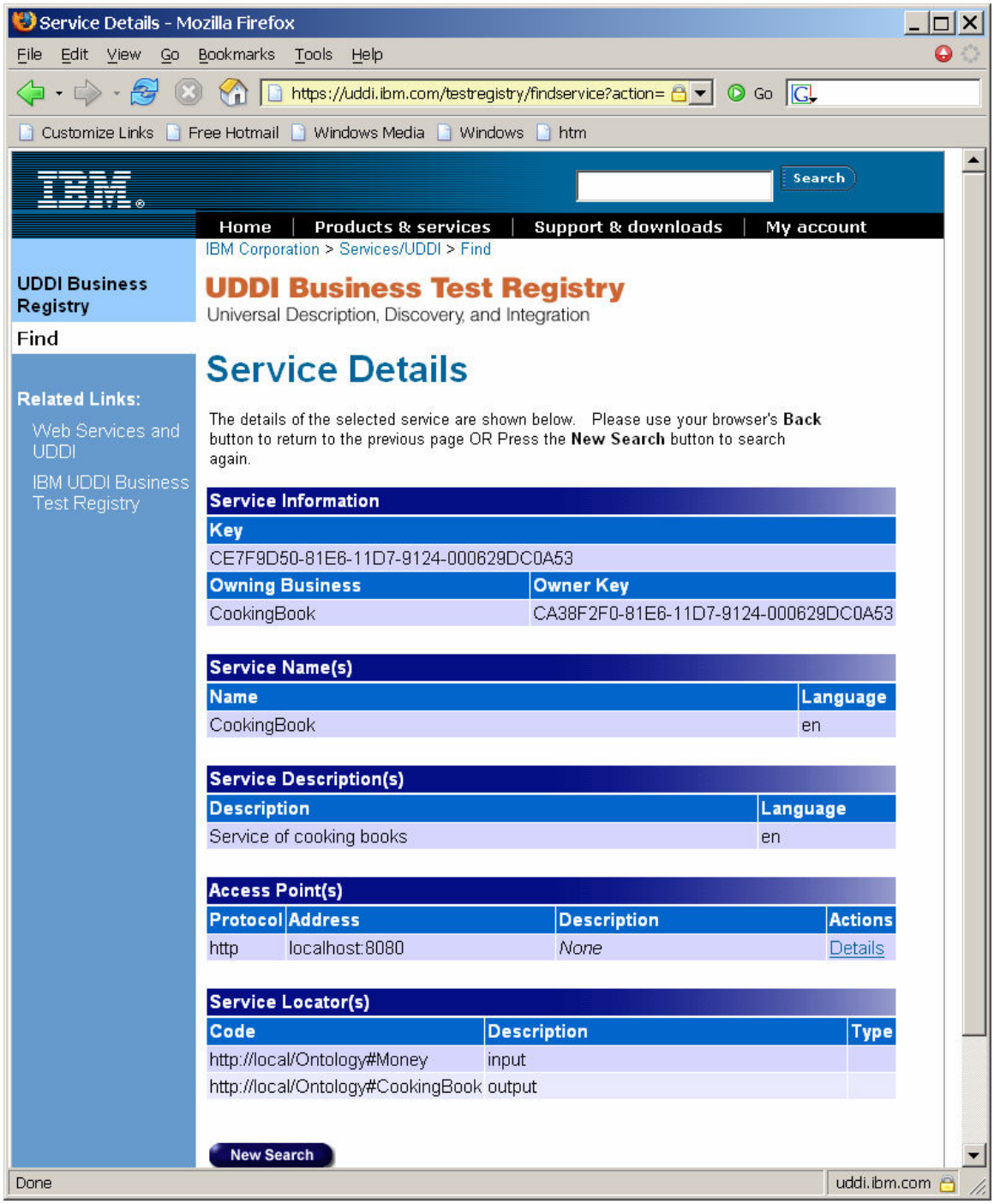

Fig. 14. details of the semantic Web service

```
[~/public_html/thesis/code/myUDDIWalker][10:40am] java myUDDIWalker.Main

connection to registry is created...
got registry service, query manager, and life cycle manager...
services who used some input_tModel:
service key:CE7F9D50-81E6-11D7-9124-000629DC0A53
services who used some output_tModel:
service key:CE7F9D50-81E6-11D7-9124-000629DC0A53
services who used both input/output_tModel:
service key:CE7F9D50-81E6-11D7-9124-000629DC0A53
------------------------------------------------------------
service detail:
service name: CookingBook
service key:CE7F9D50-81E6-11D7-9124-000629DC0A53
classification name: com.sun.xml.registry.uddi.infomodel.InternationalStringImpl@199939
classification name: com.sun.xml.registry.uddi.infomodel.InternationalStringImpl@9a9b65
[~/public_html/thesis/code/myUDDIWalker][10:40am]
```

Fig. 15. results obtained by using the current UDDI crawler

5.5 Examples of Matching Algorithm's Results

After the SWS repository is created, the broker can search for candidate Web services in it. To do so, the service request has to be also expressed in OWL-S language, and this request has to be read and processed by the matchmaking engine. This section presents the details of this final process of the proposed architecture.

5.5.1 Using OWL-S to Represent Service Requests

In order for the matchmaking engine to work, there should be a service request submitted to the matchmaking algorithm. As discussed in earlier sections, these service requests should also be written in OWL-S language. Figure 16 shows one of such service requests. This request is also used as example in the following sections.

```
<?xml version='1.0' encoding='ISO-8859-1'?>
```

```
<!DOCTYPE uridef[
   <!ENTITY rdf       "http://www.w3.org/1999/02/22-rdf-syntax-ns">
   <!ENTITY rdfs      "http://www.w3.org/2000/01/rdf-schema">
   <!ENTITY owl       "http://www.w3.org/2002/07/owl">
   <!ENTITY service   "http://www.daml.org/services/owl-s/1.0/Service.owl">
   <!ENTITY process   "http://www.daml.org/services/owl-s/1.0/Process.owl">
   <!ENTITY profile   "http://www.daml.org/services/owl-s/1.0/Profile.owl">
   <!ENTITY actor     "http://www.daml.org/services/owl-s/1.0/ActorDefault.owl">
   <!ENTITY domainOnt
             "http://tinman.cs.gsu.edu/~lyu2/thesis/SimpleBioOntology.owl">
   <!ENTITY DEFAULT   "http://tinman.cs.gsu.edu/~lyu2/thesis/request2.owl">
]>

<rdf:RDF
   xmlns:rdf=       "&rdf;#"
   xmlns:rdfs=      "&rdfs;#"
   xmlns:owl=       "&owl;#"
   xmlns:service=   "&service;#"
   xmlns:process=   "&process;#"
   xmlns:profile=   "&profile;#"
   xmlns:actor=     "&actor;#"
   xmlns:domainOnt= "&domainOnt;#"
   xmlns=           "&DEFAULT;#">

   <owl:Ontology about="">
      <owl:imports rdf:resource="&rdf;" />
      <owl:imports rdf:resource="&rdfs;" />
      <owl:imports rdf:resource="&owl;" />
      <owl:imports rdf:resource="&service;" />
      <owl:imports rdf:resource="&profile;" />
      <owl:imports rdf:resource="&process;" />
      <owl:imports rdf:resource="&actor;" />
      <owl:imports rdf:resource="&domainOnt;" />
   </owl:Ontology>

   <profile:Profile rdf:ID="BioInformatics_ws_req2">

      <profile:hasInput rdf:resource="#input1-protein"/>
      <profile:hasInput rdf:resource="#input2-protein"/>
      <profile:hasOutput rdf:resource="#output1-alignment"/>

   </profile:Profile>

   <process:Input rdf:ID="input1-protein">
      <process:parameterType rdf:resource="&domainOnt;#Protein"/>
   </process:Input>

   <process:Input rdf:ID="input2-protein">
      <process:parameterType rdf:resource="&domainOnt;#Protein"/>
   </process:Input>

   <process:UnConditionalOutput rdf:ID="output1-alignment">
      <process:parameterType rdf:resource="&domainOnt;#GlobalAlignment"/>
   </process:UnConditionalOutput>

</rdf:RDF>
```

Fig. 16. an example of a service request written in OWL-S

It is clear from this service request that the user is looking for a service that will accept two `Protein` sequences as input and output the `GlobalAlignment` back to the user. This example will be used in the next few sections to show the work done by the matching algorithm.

5.5.2 Using `Jean` APIs to Understand OWL-S Profiles and Make References

As stated previously, this prototyping system is developed using Java. To "understand" the ontology and OWL-S profiles (including both service descriptions and service requests), `Jean` is heavily used – part of its APIs is designed and implemented to make ontologies and OWL-S profiles understandable to machines. The development of the OWL-S crawler discussed in the previous section shows the details about using `Jean` to parse OWL-S profiles. The logic used to understand OWL-S profiles is quite similar to that used to parse the domain specific ontology file. Appendix B is the result of using `Jean` to parse the example ontology, including all the classes that are defined in this ontology and the relations between these classes.

The other part of its APIs is an inference engine, which is proved to be very useful in the development of the matchmaking algorithm. To see this, consider the following typical scenario where inference is needed. Suppose one is searching for a Web service that takes `Protein` as the input. The search agent now finds one service that accepts `AminoAcidSequence` as the input. In this case, with the help from the inference engine, the agent can quickly realize that `Protein` is in fact is a sub class of `AminoAcidSequence`, therefore, the current web service is a potentially qualified candidate. In fact, this is the most frequently used feature when the matchmaking engine in this search is developed.

`Jena` already has the inference engine embedded into its APIs. To activate this inference engine, the `OntModelSpec.OWL_MEM_RULE_INF` parameter should be used. On the other hand, `OntModelSpec.OWL_MEM` will disable the inference engine. For more information, see `Jena`'s office site given in [41].

5.5.3 Using the Matching Algorithm to Find Candidate Set

This section presents one example of using the matchmaking engine to find the candidate service set. When a service request is received by the broker, it will invoke the matchmaking algorithm to search in the repository to find potential candidates. Figure 17 presents a screen shot of using this algorithm. The service request is shown in Figure 16.

Clearly, the matchmaking engine first makes a connection with the SWS repository, and then it reads in the service request file. In this example, the user is looking for a service that will accept two `Protein` sequences as input parameters and output the `GlobalAlignment` back to the user. The engine ignores several candidates since 1) they do not have the correct number of input parameters (without the need to further check the output), 2) they use different domain ontologies.

In this example, the engine presents 3 candidates. The first one has an exact match for the output, but it is using `Sequence` as the input, since Sequence is a super class of `Protein`, this match is a `Level-1` match. The second one is another `Level-1` match: the output `Alignment` is again a super class of the required output `GlobalAlignment`. The same is for the last match: the output matches exactly, but `AminoAcidSequence` is a super class of `Protein`.

Fig. 17. a screen shot of the matchmaking result

CHAPTER 6

CONCLUSIONS AND FUTURE RESEARCH WORK

6.1 Conclusions and Summary of Contribution

Web services are playing an important role in business application integrations and other application fields such as bioinformatics. With the development of the related technologies, there have been more and more Web services available for use. Given this proliferation of available Web services, it is therefore crucial for the service consumers to discover and select the desired services in an efficient and even automated manner: only after the automated discovery and selection of the relevant Web services is accomplished, one can further discuss other highly desirable capabilities such as automated service invocation, composition and monitoring.

Given this background, this research proposed an indexation and discovery architecture for semantic Web services, together with an application example in the area of bioinformatics. In this approach, a semantic Web service repository is created and maintained by a Web crawler which collects semantic Web service descriptions from both the ontology-oriented UDDI registries as well the Web sites hosting these services. After receiving a service request, the broker in the system will invoke a matching algorithm and a candidate set will be returned to the user. Different degree of matching is also considered when the candidate set is created.

This proposed architecture is further implemented as a prototyping system in the area of bioinformatics, using Java and `Jena` APIs. The running example of the prototyping system shows that

different components of the system can work together to collect the semantic Web service profiles, build the SWS repository and finally present to the user the desired service candidates.

Some of the contributions of this research work are summarized in Section 3.3, which covers the main benefits of the proposed architecture. To briefly reiterate, these benefits include the following:

- More comprehensive coverage

This follows from the fact that the Web crawler visits not only the public/private UDDI registries, but also the Web to find all the Web sites hosting semantic Web services. Therefore, a search request submitted to the system is satisfied by not only searching the UDDI registries, but also the potentially a large number of Web sites.

- More flexibility to service publisher

Besides the option to register the service into UDDI registry (and insert relevant UDDI tModels to add the semantic descriptions), the publisher can also elect to directly publish the service profiles on their Web sites without adding any entry into the UDDI registries.

- More up-to-date repository

A centralized UDDI registry always suffers form the possibility of being obsolete. However, the SWS repository can be updated periodically.

- More transparency to the client

In the proposed architecture, the broker hides the complexity of the matching algorithms. When a particular service is requested, the effort on the client side can be minimized.

Besides the above contributions, the following can be further summarized here:

- The proposed architecture can be easily extended to include more desirable features.

For instance, more powerful matching algorithms can be used to replace the current one, and QoS model can be built on top of the matching algorithm to further “screening” the candidate set (details can be found in [21]).

- As a proof of concept, this research provides a concrete implementation of a working semantic Web service discovery system.

This includes an example ontology written in OWL, Web service description markup using OWL-S language, semantic Web service repository created by a Web crawler, and a matchmaking engine. At the current stage of this research area and to our best knowledge, there are not many concrete examples (even prototypes) that have all the necessary components working together to implement the discovery of semantic Web services.

- As a proof of concept, this research also provides an example in the area of Bioinformatics to illustrate the value that can be added by using semantic Web services in these important application fields.

As discussed earlier, Bioinformatics involves vast amount of data and knowledge resources that are widely distributed, highly heterogeneous and diverse, yet the biological questions to be answered are computationally quite complex. This makes the Web based tools the ideal candidate. The implemented system in this research, although a prototype, provides a concrete example of how the semantic Web services can add more value to Web based application for Bioinformatics methods and discoveries.

6.2 Future Research Work

Possible future research work is discussed in this last Section. This includes the possible improvements to the current prototype system and potential new research directions.

To improve the current prototyping system, the following can be considered:

- More sophisticated matching algorithms can be designed and implemented into the system.

For example, preconditions and effects can be considered in the matchmaking algorithm. The description of preconditions and effects can be added easily to the OWL-S profiles using the OWL-S

`ServiceProfile` upper ontology. Other possible improvements, such as letting the service requester express special "user-defined matching" rules [23], can also be explored. Another example is to introduce QoS models into the matchmaking engine, detailed discuss about this approach and implementation can be found in [21].

- More controls and improvements to the Web and UDDI crawler.

The performance of the proposed system not only depends on the matching algorithm, but also the Web/UDDI crawler: the more semantic Web service descriptions the crawler can collect, the more chance a given service request would be satisfied. Currently, the crawler may access large numbers of pages that don't contain OWL-S profile descriptions, and precious system resources may be wasted. The use of heuristics to focus on OWL-S pages may be desirable. In particular, we are looking at the so-called "weighting algorithms" – some of the examples can be found in Expose [40].

Potential new research directions can include the following:

- Automatic service invocation.

Automatic invocation of the service can be considered as a future research direction. The current indexation and discovery structure focus mainly on the discovery of the services, and the next logic step would be to further let the system handle the service binding and execute the services on behalf of the service requester. More specifically, a client will send the service parameters together with the service discovery request, and receive the final results if candidate services have been found. For a detailed introduction and discussion of service invocation, see [42].

- Automatic service composition

Following directly from the automatic invocation, another research direction is the composition of Web services. Composition is the process of selecting, combining and executing Web services to achieve a user's object. Human beings perform manual Web service composition by exploring their

knowledge of what a Web service does, as well as information provided on the service's Web pages, in order to execute a collection of services to achieve some objective. To automate Web service composition, al this information must be encoded explicitly in an unambiguous computer interpretable from. This issue closely related to the service discovery and also seems to be a crucial feature in the Web of the future. For a more comprehensive discussion of service composition, see [6].

REFERENCE

[1]. T. Berners-Lee, J. Hendler, O. Lassila, “The Semantic Web”, Scientific American, 284(5), 2001, 34-43

[2]. F. Curbera, W. Nagy, S. Weerawarana, “Web services: Why and how”, Workshop on Object-Oriented Web Services – OOPSLA 2001, Tampa, FL, 2001

[3] M. Gudgin, M. Hadley, N. Mendelsohn, J. Moreau, H. Nielsen, “SOAP Version 1.2 Part 1” , W3C Working Draft, June 2003, Available at http://www.w3.org/TR/2003/REC-soap12-part1-20030624/

[4] Chinnici, R., Gudgin, M., Moreau, J. and Weerawarana, S. “Web Services Description Language (WSDL) Version 1.2", W3C Working Draft, January 2003, Available at http://www.w3.org/TR/2003/WD-wsdl12-20030124/

[5] Universal Description, Discovery and Integration: UDDI Technical White paper. 2000. http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf

[6] Sheila A. McIlraith and David L. Martin. “Bringing semantics to web services”, *IEEE Intelligent Systems*, 18:90–93, January/February 2003.

[7] J. Hendler, “Agents and the semantic web”, IEEE Intelligent Systems 16(2), March/April, 2001

[8] K. Sivashanmugam, K. Verma, A. Sheth and J. Miller, “Adding Semantics to Web Services Standards”, International Conference on Web Services (ICWS’03), pp. 395-401, 2003

[9] Ankolenkar, A., Burstein, M., Hobbs, J.R., Lassila, O., Martin, D.L., McDermott, D., McIlraith, S.A., Narayanan, S., Paolucci, M., Payne T.R., and Sycara, K. The DAML Services Coalition, "DAML-S: Web Service Description for the Semantic Web", The First International Semantic Web Conference (ISWC), Sardinia (Italy), June, 2002.

[10]. M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara. Importing the semantic web in uddi. In Proceedings of E-Services and the Semantic Web Workshop, 2002. http://citeseer.ist.psu.edu/article/paolucci02importing.html

[11] The OWL Services Coalition. OWL-S: Semantic Markup for Web Services. Technical report, http://www.daml.org/services/, 2004

[12] Naveen Srinivasan, Massimo Paolucci and Katia Sycara, "An Efficient Algorithm for OWL-S based Semantic Search in UDDI" First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004) 6-9, 2004, San Diego, California, USA.

[13] Akkiraju, R., Goodwin, R., Doshi, P., and Roeder S., "A method for semantically enhancing the service discovery capabilities of UDDI", Proceedings of the Workshop on Information Integration on the Web, 87-92, August 2003

[14] Verma, K., Sivashanmugam K., Sheth A., Patil A., "METEOR-S WSDL: A scalable P2P infrastructure of registries for semantic publication and discovery of Web services", Journal of Information Technology and Management, in print.

[15] IRS-II A Framework and Infrastructure for Semantic Web Services Enrico Motta, John Dominigue, Liliana Cabral, Mauro Gaspari, 2nd International Semantic Web Conference 2003 (ISWC 2003).

[16] B. Omelayenko, M. Crubezy, D. Fensel, R. Benjamins, UPML: the Language and Tool Support for Making the Semantic Web Alive, in Spinning the Semantic Web: Bringing the WWW to Its Full Potential, MIT Press, 2003, pp.141-170

[17] U. Thaden, W. Siberski, and W. Nejdl, A Semantic Web Based Peer-to-Peer Service Registry Network, Technical Report, Learning Lab Lower Saxony, 2003

[18] Kaarthik Sivashanmugam, Kunal Verma, Amit P. Sheth: Discovery of Web Services in a Federated Registry Environment. ICWS 2004: 270-278

[19] Chen Zhou, Liang-Tien Chia, Bilhanan Silverajan, Bu-Sung Lee: UX- An Architecture Providing QoS-Aware and Federated Support for UDDI. ICWS 2003: 171-176

[20] Essafi, Tarek; Dorta, Neilze; Seret, Dominique; Makpangou, Mesaac (France): "A Scalable Peer-To-Peer Approach to Service Discovery Using Ontology", The 9th World Multiconference on Systemics, Cybernetics and Informatics, July 10 - 13, 2005 Orlando, Florida, USA

[21]. Wang, H., Smarandache, F., Zhang, Y. and Sunderraman R., "Interval Neutrosophic Sets and Logic: Theory and Applications in Computing", HEXIS, Neutrosophic Book Series, No.5, 2005

[22] M. Paolucci, T. Kawmura, T. Payne and K. Sycara. Semantic Matching of Web Services Capabilities. In First Int. Semantic Web Conf., To appear 2002. http://citeseer.ist.psu.edu/paolucci02semantic.html

[23] Michael C. Jaeger, Gregor Rojec-Goldmann, Christoph Liebetruth, Gero Muhl and Kurt Geihs: "*Ranked Matching for Service Descriptions Using OWL-S",* KiVS 2005: 91-102

[24] D. Trastour, C. Bartolini and J. Gonzalez-Castillo, "A Semantic Web Approach to Service Description for Matchmaking of Services", Proceedings of the International Semantic Web Working Symposium (SWWS), 2001

[25] Bansal S., Vidal J. M., "Matchmaking of web services based on the DAML-S service model", Proceedings of AAMAS'03, ACM Press, July 2003

[26] Anupriya Ankolenkar et al. DAML-S: A semantic markup language for web services. In Proceedings of 1st Semantic Web Working Symposium (SWWS' 01), pages 441–430, Stanford, USA, August 2001. Stanford University.

[27] Booth D. et al, "Web services architecture", Technical report, W3C, http://www.w3.org/TR/ws-arch/, 2004

[28] Mei Kobayashi and Koichi Takeda. Information retrieval on the web. *ACM Computing Surveys*, (2):144–173, June 2000.

[29] Frank Manola et al. "RDF Primer: Technical report", W3C, http://www.w3.org/TR/rdf-primer/, 2004.

[30] Brickley, D; Guha, R. V., "Resource Description Framework (RDF) Schema Specification 1.0", W3C Candidate Recommendation, March 2000; http://www.w3.org/TR/rdf-schema/.

[31] Hendler, J., McGuinness, D. L.; "The DARPA Agent Markup Language", IEEE Intelligent Systems, Vol.16, No. 6, Jan./Feb., 2000, pp. 67-74

[32] Deborah L. McGuinness and Frank van Harmelen. "Owl web ontology language overview". Technical report, W3C, http://www.w3.org/TR/owl-features/, 2004.

[33] The OWL Services Coalition. OWL-S Example Description for Bravo Air. Technical report, http://www.daml.org/services/owl-s/1.0/BravoAirProfile.owl.

[34] "Describing Web Services using OWL-S and WSDL", DAML-S Coalition working document, October 2003

[35] Robert Stevens, Carole A. Goble, and Sean Bechhofer. *Ontology-based knowledge representation for bioinformatics*. Briefings in Bioinformatics, 1(4):398--414, Nov 2000.

[36] McCray A. "An upper level ontology for the biomedical domain", Comp Functional Genomics 2003; 4: 80-84.

[37] Chris Wroe, Robert Stevens, Carole A. Goble, Angus Roberts, R. Mark Greenwood: "A Suite of Daml+Oil Ontologies to Describe Bioinformatics Web Services and Data", *International Journal of Cooperative Information Systems,* 12(2): 197-224 (2003)

[38] "OWL-S/UDDI Matchmaker Web Interface" Project, the Software Agents Group of the Robotics Institute at Carnegie Mellon University, available at http://www.daml.ri.cmu.edu/matchmaker/webinterface.htm

[39] DAML crawler, available at http://www.daml.org/crawler/

[40] Expose, A SHOE crawler, available at http://www.cs.umd.edu/projects/plus/SHOE/Expose.html

[41] `Jena` Semantic Web Framework, available at http://`Jena`.sourceforge.net/

[42] S. Lu, M. Dong, and F. Fotouhi. The semantic web: Opportunities and challenges for next-generation web applications. International Journal of Information Research, 7(4), 2002. Special Issue on the Semantic Web. http://citeseer.csail.mit.edu/lu02semantic.html

APPENDICES

A. A small ontology in the area of Bioinformatics

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<!-- this is a experimental ontology in the area of bioinformatics -->
<!-- for my master research in Dept of Computer Science -->
<!-- created by Liyang Yu, August 6, 2005 -->

<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:owl="http://www.w3.org/2002/07/owl#"
         xmlns="http://www.yuchen.net/#"
         xmlns:simpleBioOntology="http://www.yuchen.net/#"
         xml:base="http://www.yuchen.net/">

<!--
     xml:base identifies the default namespace used in this ontology
     model, sometimes, it is used to identify ewhere this xml document
     is placed. also notice that this base URL does not have the "#",
     therefore, all the concepts that are in this namespace will have
     to have prefixed by a "#".

     for example, if you want to access class Protein using Jena, the
     namespace we should use is

     http://tinman.cs.gsu.edu/~lyu2/thesis/SimpleBioOntology

     the full class name will be:

     http://tinman.cs.gsu.edu/~lyu2/thesis/SimpleBioOntology#Protein

-->

<!-- add label and comments to these classes and properties -->
<!-- so we can remember them -->

     <!-- ################### -->
     <!--  top level classes  -->
     <!-- ################### -->

     <owl:Class rdf:ID="Structure">
        <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
     </owl:Class>

     <owl:Class rdf:ID="SequenceCharacteristics">
        <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
     </owl:Class>

     <owl:Class rdf:ID="Alignment">
        <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
     </owl:Class>

     <owl:Class rdf:ID="Function">
        <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
     </owl:Class>

     <owl:Class rdf:ID="Chromosome">
        <rdfs:label>chromosome</rdfs:label>
        <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
     </owl:Class>
```

```
<owl:Class rdf:ID="Gene">
   <rdfs:label>chromosome</rdfs:label>
   <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
</owl:Class>

<owl:Class rdf:ID="ProteinName">
   <rdfs:label>proteinName</rdfs:label>
   <rdfs:comment>the name of a given protein structure</rdfs:comment>
   <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
</owl:Class>

<!-- ################### -->
<!--  all sub classes    -->
<!-- ################### -->

<owl:Class rdf:ID="BiologicalStructure">
   <rdfs:subClassOf rdf:resource="#Structure"/>
</owl:Class>

<owl:Class rdf:ID="Sequence">
   <rdfs:subClassOf rdf:resource="#BiologicalStructure"/>
</owl:Class>

<owl:Class rdf:ID="SecondaryStructure">
   <rdfs:subClassOf rdf:resource="#BiologicalStructure"/>
</owl:Class>

<owl:Class rdf:ID="TertiaryStructure">
   <rdfs:subClassOf rdf:resource="#BiologicalStructure"/>
</owl:Class>

<!-- this identifies a location within a given sequence -->
<!-- so later on, we can use this to do subsequence operations -->
<owl:Class rdf:ID="SequenceLocation">
   <rdfs:subClassOf rdf:resource="#SequenceCharacteristics"/>
</owl:Class>

<!-- this identifies the "similarity" of two sequences -->
<owl:Class rdf:ID="SequenceSimilarity">
   <rdfs:subClassOf rdf:resource="#SequenceCharacteristics"/>
</owl:Class>

<!-- this identifies the length of a given sequence -->
<!-- we can use this to do sequenceLength opertions -->
<owl:Class rdf:ID="SequenceLength">
   <rdfs:subClassOf rdf:resource="#SequenceCharacteristics"/>
</owl:Class>

<owl:Class rdf:ID="GlobalAlignment">
   <rdfs:subClassOf rdf:resource="#Alignment"/>
</owl:Class>

<owl:Class rdf:ID="LocalAlignment">
   <rdfs:subClassOf rdf:resource="#Alignment"/>
</owl:Class>

<owl:Class rdf:ID="DNA">
   <rdfs:label>dna</rdfs:label>
   <rdfs:subClassOf rdf:resource="#Sequence"/>
</owl:Class>

<owl:Class rdf:ID="MessengerRNA">
   <rdfs:label>dna</rdfs:label>
   <rdfs:subClassOf rdf:resource="#Sequence"/>
</owl:Class>

<owl:Class rdf:ID="AminoAcidSequence">
   <rdfs:label>dna</rdfs:label>
   <rdfs:subClassOf rdf:resource="#Sequence"/>
</owl:Class>
```

```
<owl:Class rdf:ID="Polypeptide">
   <rdfs:label>dna</rdfs:label>
   <rdfs:subClassOf rdf:resource="#AminoAcidSequence"/>
</owl:Class>

<owl:Class rdf:ID="Protein">
   <rdfs:label>dna</rdfs:label>
   <rdfs:subClassOf rdf:resource="#Polypeptide"/>
</owl:Class>

<owl:Class rdf:ID="PrimaryStructure">
   <owl:equivalentClass rdf:resource="#Polypeptide"/>
</owl:Class>

<owl:Class rdf:ID="AlphaHelix">
   <rdfs:subClassOf rdf:resource="#SecondaryStructure"/>
</owl:Class>

<owl:Class rdf:ID="BetaSheet">
   <rdfs:subClassOf rdf:resource="#SecondaryStructure"/>
</owl:Class>

<owl:Class rdf:ID="TertiaryX">
   <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
</owl:Class>

<owl:Class rdf:ID="TertiaryY">
   <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
</owl:Class>

<owl:Class rdf:ID="TertiaryZ">
   <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
</owl:Class>

<owl:Class rdf:ID="BiologicalFunction">
   <rdfs:subClassOf rdf:resource="#Function"/>
</owl:Class>

<owl:Class rdf:ID="ProteinFunction">
   <rdfs:subClassOf rdf:resource="#BiologicalFunction"/>
</owl:Class>

<!-- ###################### -->
<!--  property definition  -->
<!-- ###################### -->

<owl:ObjectProperty rdf:ID="organized-into">
   <rdfs:domain rdf:resource="#DNA"/>
   <rdfs:range rdf:resource="#Chromosome"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="strings-of">
   <rdfs:domain rdf:resource="#Chromosome"/>
   <rdfs:range rdf:resource="#Gene"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="transcribed-from">
   <rdfs:domain rdf:resource="#MessengerRNA"/>
   <rdfs:range rdf:resource="#DNA"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="translated-from">
   <rdfs:domain rdf:resource="#AminoAcidSequence"/>
   <rdfs:range rdf:resource="#MessengerRNA"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="has-name">
   <rdfs:domain rdf:resource="#Protein"/>
   <rdfs:range  rdf:resource="#ProteinName"/>
</owl:ObjectProperty>
```

```
<owl:DatatypeProperty rdf:ID="protein-name-type">
   <rdfs:domain rdf:resource="#ProteinName" />
   <rdfs:range  rdf:resource="http://www.w3.org/2001/XMLSchema#string" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="sequence-type">
   <rdfs:domain rdf:resource="#Sequence" />
   <rdfs:range  rdf:resource="http://www.w3.org/2001/XMLSchema#string" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="sequenceLocation-type">
   <rdfs:domain rdf:resource="#SequenceLocation" />
   <rdfs:range  rdf:resource="http://www.w3.org/2001/XMLSchema#integer" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="sequenceLength-type">
   <rdfs:domain rdf:resource="#SequenceLength" />
   <rdfs:range  rdf:resource="http://www.w3.org/2001/XMLSchema#integer" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="sequenceSimilarity-type">
   <rdfs:domain rdf:resource="#SequenceSimilarity" />
   <rdfs:range  rdf:resource="http://www.w3.org/2001/XMLSchema#float" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="x-type">
   <rdfs:domain rdf:resource="#TertiaryX" />
   <rdfs:range  rdf:resource="http://www.w3.org/2001/XMLSchema#float" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="y-type">
   <rdfs:domain rdf:resource="#TertiaryY" />
   <rdfs:range  rdf:resource="http://www.w3.org/2001/XMLSchema#float" />
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="z-type">
   <rdfs:domain rdf:resource="#TertiaryZ" />
   <rdfs:range  rdf:resource="http://www.w3.org/2001/XMLSchema#float" />
</owl:DatatypeProperty>

<owl:ObjectProperty rdf:ID="start-location">
   <rdfs:domain rdf:resource="#Sequence"/>
   <rdfs:range  rdf:resource="#SequenceLocation"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="end-location">
   <rdfs:domain rdf:resource="#Sequence"/>
   <rdfs:range  rdf:resource="#SequenceLocation"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="length">
   <rdfs:domain rdf:resource="#Sequence"/>
   <rdfs:range  rdf:resource="#SequenceLength"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="similarity">
   <rdfs:domain rdf:resource="#Sequence"/>
   <rdfs:range  rdf:resource="#SequenceSimilarity"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="alignment">
   <rdfs:domain rdf:resource="#Sequence"/>
   <rdfs:range  rdf:resource="#Alignment"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="has-primary-structure">
   <rdfs:domain rdf:resource="#Protein"/>
   <rdfs:range  rdf:resource="#PrimaryStructure"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="has-secondary-structure">
```

```
      <rdfs:domain rdf:resource="#Protein"/>
      <rdfs:range  rdf:resource="#SecondaryStructure"/>
   </owl:ObjectProperty>

   <owl:ObjectProperty rdf:ID="has-tertiary-structure">
      <rdfs:domain rdf:resource="#Protein"/>
      <rdfs:range  rdf:resource="#TertiaryStructure"/>
   </owl:ObjectProperty>

   <owl:ObjectProperty rdf:ID="has-x">
      <rdfs:domain rdf:resource="#TertiaryStructure"/>
      <rdfs:range  rdf:resource="#TertiaryX"/>
   </owl:ObjectProperty>

   <owl:ObjectProperty rdf:ID="has-y">
      <rdfs:domain rdf:resource="#TertiaryStructure"/>
      <rdfs:range  rdf:resource="#TertiaryY"/>
   </owl:ObjectProperty>

   <owl:ObjectProperty rdf:ID="has-z">
      <rdfs:domain rdf:resource="#TertiaryStructure"/>
      <rdfs:range  rdf:resource="#TertiaryZ"/>
   </owl:ObjectProperty>

   <owl:ObjectProperty rdf:ID="has-function">
      <rdfs:domain rdf:resource="#Protein"/>
      <rdfs:range rdf:resource="#ProteinFunction"/>
   </owl:ObjectProperty>

</rdf:RDF>
```

B. Classes and Their Relations Parsed by Using Jena APIs

```
===== all the classes that are defined ===

 --- classes in this ontology file ---
http://www.w3.org/2002/07/owl#Nothing
http://www.w3.org/2002/07/owl#Thing
http://www.yuchen.net/#GlobalAlignment
http://www.yuchen.net/#Function
http://www.yuchen.net/#Protein
http://www.yuchen.net/#ProteinFunction
http://www.yuchen.net/#BiologicalStructure
http://www.yuchen.net/#TertiaryY
http://www.yuchen.net/#Alignment
http://www.yuchen.net/#TertiaryX
http://www.yuchen.net/#SequenceLocation
http://www.yuchen.net/#AminoAcidSequence
http://www.yuchen.net/#Chromosome
http://www.yuchen.net/#Structure
http://www.yuchen.net/#SequenceLength
http://www.yuchen.net/#ProteinName
http://www.yuchen.net/#Gene
http://www.yuchen.net/#LocalAlignment
http://www.yuchen.net/#MessengerRNA
http://www.yuchen.net/#SequenceSimilarity
http://www.yuchen.net/#TertiaryStructure
http://www.yuchen.net/#Sequence
http://www.yuchen.net/#TertiaryZ
http://www.yuchen.net/#Polypeptide
http://www.yuchen.net/#PrimaryStructure
http://www.yuchen.net/#BiologicalFunction
http://www.yuchen.net/#BetaSheet
http://www.yuchen.net/#SecondaryStructure
http://www.yuchen.net/#DNA
http://www.yuchen.net/#SequenceCharacteristics
http://www.yuchen.net/#AlphaHelix
http://www.w3.org/2001/XMLSchema#short
http://www.w3.org/2001/XMLSchema#int
http://www.w3.org/2001/XMLSchema#long
http://www.w3.org/2001/XMLSchema#byte
http://www.w3.org/2001/XMLSchema#unsignedInt
http://www.w3.org/2001/XMLSchema#unsignedByte
http://www.w3.org/2001/XMLSchema#unsignedLong
http://www.w3.org/2001/XMLSchema#nonNegativeInteger
http://www.w3.org/2001/XMLSchema#unsignedShort
http://www.w3.org/1999/02/22-rdf-syntax-ns#List
http://www.w3.org/2000/01/rdf-schema#Class
http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement
http://www.w3.org/2002/07/owl#Property
http://www.w3.org/2001/XMLSchema#string
http://www.w3.org/1999/02/22-rdf-syntax-ns#Property
http://www.w3.org/2002/07/owl#Ontology
http://www.w3.org/2000/01/rdf-schema#Resource
http://www.w3.org/2001/XMLSchema#integer
http://www.w3.org/2002/07/owl#Restriction
http://www.w3.org/2001/XMLSchema#float
http://www.w3.org/2002/07/owl#Class
http://www.w3.org/2000/01/rdf-schema#Literal
http://www.w3.org/2001/XMLSchema#date
http://www.w3.org/2001/XMLSchema#boolean
http://www.w3.org/2001/XMLSchema#decimal
http://www.w3.org/2001/XMLSchema#duration
http://www.w3.org/2001/XMLSchema#nonPositiveInteger
http://www.w3.org/2001/XMLSchema#dateTime
http://www.w3.org/2001/XMLSchema#time
 ---------- end of classes -----------
===== class structure in the model ====
```

```
Class: owl:Nothing
  is a sub-class of Class: owl:Thing
  is a sub-class of Class: rdfs:Resource
Class: owl:Thing
  is a sub-class of Class: rdfs:Resource
  is a super-class of Class: simpleBioOntology:AlphaHelix
  is a super-class of Class: owl:Nothing
  is a super-class of Class: simpleBioOntology:TertiaryStructure
  is a super-class of Class: simpleBioOntology:Sequence
  is a super-class of Class: simpleBioOntology:SequenceLength
  is a super-class of Class: simpleBioOntology:AminoAcidSequence
  is a super-class of Class: simpleBioOntology:BiologicalFunction
  is a super-class of Class: simpleBioOntology:SequenceLocation
  is a super-class of Class: simpleBioOntology:DNA
  is a super-class of Class: simpleBioOntology:BetaSheet
  is a super-class of Class: simpleBioOntology:LocalAlignment
  is a super-class of Class: simpleBioOntology:ProteinFunction
  is a super-class of Class: simpleBioOntology:PrimaryStructure
  is a super-class of Class: simpleBioOntology:Polypeptide
  is a super-class of Class: simpleBioOntology:SequenceSimilarity
  is a super-class of Class: simpleBioOntology:GlobalAlignment
  is a super-class of Class: simpleBioOntology:SecondaryStructure
  is a super-class of Class: simpleBioOntology:BiologicalStructure
  is a super-class of Class: simpleBioOntology:Protein
  is a super-class of Class: simpleBioOntology:MessengerRNA
  is a super-class of Class: simpleBioOntology:Chromosome
  is a super-class of Class: simpleBioOntology:Alignment
  is a super-class of Class: simpleBioOntology:TertiaryY
  is a super-class of Class: simpleBioOntology:TertiaryX
  is a super-class of Class: simpleBioOntology:Gene
  is a super-class of Class: simpleBioOntology:Structure
  is a super-class of Class: simpleBioOntology:ProteinName
  is a super-class of Class: simpleBioOntology:SequenceCharacteristics
  is a super-class of Class: simpleBioOntology:TertiaryZ
  is a super-class of Class: simpleBioOntology:Function
Class: simpleBioOntology:GlobalAlignment
  is a sub-class of Class: rdfs:Resource
  is a sub-class of Class: owl:Thing
  is a sub-class of Class: simpleBioOntology:Alignment
Class: simpleBioOntology:Function
  is a sub-class of Class: rdfs:Resource
  is a sub-class of Class: owl:Thing
  is a super-class of Class: simpleBioOntology:BiologicalFunction
  is a super-class of Class: simpleBioOntology:ProteinFunction
Class: simpleBioOntology:Protein
  is a sub-class of Class: rdfs:Resource
  is a sub-class of Class: owl:Thing
  is a sub-class of Class: simpleBioOntology:Polypeptide
  is a sub-class of Class: simpleBioOntology:Sequence
  is a sub-class of Class: simpleBioOntology:BiologicalStructure
  is a sub-class of Class: simpleBioOntology:AminoAcidSequence
  is a sub-class of Class: simpleBioOntology:Structure
  is a sub-class of Class: simpleBioOntology:PrimaryStructure
Class: simpleBioOntology:ProteinFunction
  is a sub-class of Class: owl:Thing
  is a sub-class of Class: rdfs:Resource
  is a sub-class of Class: simpleBioOntology:BiologicalFunction
  is a sub-class of Class: simpleBioOntology:Function
Class: simpleBioOntology:BiologicalStructure
  is a sub-class of Class: rdfs:Resource
  is a sub-class of Class: owl:Thing
  is a sub-class of Class: simpleBioOntology:Structure
  is a super-class of Class: simpleBioOntology:TertiaryStructure
  is a super-class of Class: simpleBioOntology:Sequence
  is a super-class of Class: simpleBioOntology:SecondaryStructure
  is a super-class of Class: simpleBioOntology:BetaSheet
  is a super-class of Class: simpleBioOntology:AlphaHelix
  is a super-class of Class: simpleBioOntology:PrimaryStructure
  is a super-class of Class: simpleBioOntology:DNA
  is a super-class of Class: simpleBioOntology:Protein
  is a super-class of Class: simpleBioOntology:Polypeptide
```

```
  is a super-class of Class: simpleBioOntology:AminoAcidSequence
  is a super-class of Class: simpleBioOntology:MessengerRNA
Class: simpleBioOntology:TertiaryY
  is a sub-class of Class: rdfs:Resource
  is a sub-class of Class: owl:Thing
Class: simpleBioOntology:Alignment
  is a sub-class of Class: rdfs:Resource
  is a sub-class of Class: owl:Thing
  is a super-class of Class: simpleBioOntology:LocalAlignment
  is a super-class of Class: simpleBioOntology:GlobalAlignment
Class: simpleBioOntology:TertiaryX
  is a sub-class of Class: rdfs:Resource
  is a sub-class of Class: owl:Thing
Class: simpleBioOntology:SequenceLocation
  is a sub-class of Class: owl:Thing
  is a sub-class of Class: rdfs:Resource
  is a sub-class of Class: simpleBioOntology:SequenceCharacteristics
Class: simpleBioOntology:AminoAcidSequence
  is a sub-class of Class: owl:Thing
  is a sub-class of Class: rdfs:Resource
  is a sub-class of Class: simpleBioOntology:Sequence
  is a sub-class of Class: simpleBioOntology:BiologicalStructure
  is a sub-class of Class: simpleBioOntology:Structure
  is a super-class of Class: simpleBioOntology:Polypeptide
  is a super-class of Class: simpleBioOntology:PrimaryStructure
  is a super-class of Class: simpleBioOntology:Protein
Class: simpleBioOntology:Chromosome
  is a sub-class of Class: rdfs:Resource
  is a sub-class of Class: owl:Thing
Class: simpleBioOntology:Structure
  is a sub-class of Class: rdfs:Resource
  is a sub-class of Class: owl:Thing
  is a super-class of Class: simpleBioOntology:BiologicalStructure
  is a super-class of Class: simpleBioOntology:SecondaryStructure
  is a super-class of Class: simpleBioOntology:BetaSheet
  is a super-class of Class: simpleBioOntology:TertiaryStructure
  is a super-class of Class: simpleBioOntology:AlphaHelix
  is a super-class of Class: simpleBioOntology:PrimaryStructure
  is a super-class of Class: simpleBioOntology:DNA
  is a super-class of Class: simpleBioOntology:Protein
  is a super-class of Class: simpleBioOntology:Polypeptide
  is a super-class of Class: simpleBioOntology:Sequence
  is a super-class of Class: simpleBioOntology:AminoAcidSequence
  is a super-class of Class: simpleBioOntology:MessengerRNA
Class: simpleBioOntology:SequenceLength
  is a sub-class of Class: owl:Thing
  is a sub-class of Class: rdfs:Resource
  is a sub-class of Class: simpleBioOntology:SequenceCharacteristics
Class: simpleBioOntology:ProteinName
  is a sub-class of Class: rdfs:Resource
  is a sub-class of Class: owl:Thing
Class: simpleBioOntology:Gene
  is a sub-class of Class: rdfs:Resource
  is a sub-class of Class: owl:Thing
Class: simpleBioOntology:LocalAlignment
  is a sub-class of Class: owl:Thing
  is a sub-class of Class: rdfs:Resource
  is a sub-class of Class: simpleBioOntology:Alignment
Class: simpleBioOntology:MessengerRNA
  is a sub-class of Class: rdfs:Resource
  is a sub-class of Class: owl:Thing
  is a sub-class of Class: simpleBioOntology:Sequence
  is a sub-class of Class: simpleBioOntology:BiologicalStructure
  is a sub-class of Class: simpleBioOntology:Structure
Class: simpleBioOntology:SequenceSimilarity
  is a sub-class of Class: owl:Thing
  is a sub-class of Class: rdfs:Resource
  is a sub-class of Class: simpleBioOntology:SequenceCharacteristics
Class: simpleBioOntology:TertiaryStructure
  is a sub-class of Class: owl:Thing
  is a sub-class of Class: rdfs:Resource
```

```
  is a sub-class of Class: simpleBioOntology:BiologicalStructure
  is a sub-class of Class: simpleBioOntology:Structure
Class: simpleBioOntology:Sequence
  is a sub-class of Class: owl:Thing
  is a sub-class of Class: rdfs:Resource
  is a sub-class of Class: simpleBioOntology:BiologicalStructure
  is a sub-class of Class: simpleBioOntology:Structure
  is a super-class of Class: simpleBioOntology:AminoAcidSequence
  is a super-class of Class: simpleBioOntology:DNA
  is a super-class of Class: simpleBioOntology:MessengerRNA
  is a super-class of Class: simpleBioOntology:PrimaryStructure
  is a super-class of Class: simpleBioOntology:Protein
  is a super-class of Class: simpleBioOntology:Polypeptide
Class: simpleBioOntology:TertiaryZ
  is a sub-class of Class: rdfs:Resource
  is a sub-class of Class: owl:Thing
Class: simpleBioOntology:Polypeptide
  is a sub-class of Class: simpleBioOntology:PrimaryStructure
  is a sub-class of Class: owl:Thing
  is a sub-class of Class: rdfs:Resource
  is a sub-class of Class: simpleBioOntology:AminoAcidSequence
  is a sub-class of Class: simpleBioOntology:Sequence
  is a sub-class of Class: simpleBioOntology:BiologicalStructure
  is a sub-class of Class: simpleBioOntology:Structure
  is a super-class of Class: simpleBioOntology:PrimaryStructure
  is a super-class of Class: simpleBioOntology:Protein
Class: simpleBioOntology:PrimaryStructure
  is a sub-class of Class: simpleBioOntology:Polypeptide
  is a sub-class of Class: rdfs:Resource
  is a sub-class of Class: owl:Thing
  is a sub-class of Class: simpleBioOntology:Sequence
  is a sub-class of Class: simpleBioOntology:BiologicalStructure
  is a sub-class of Class: simpleBioOntology:AminoAcidSequence
  is a sub-class of Class: simpleBioOntology:Structure
  is a super-class of Class: simpleBioOntology:Polypeptide
  is a super-class of Class: simpleBioOntology:Protein
Class: simpleBioOntology:BiologicalFunction
  is a sub-class of Class: owl:Thing
  is a sub-class of Class: rdfs:Resource
  is a sub-class of Class: simpleBioOntology:Function
  is a super-class of Class: simpleBioOntology:ProteinFunction
Class: simpleBioOntology:BetaSheet
  is a sub-class of Class: owl:Thing
  is a sub-class of Class: rdfs:Resource
  is a sub-class of Class: simpleBioOntology:SecondaryStructure
  is a sub-class of Class: simpleBioOntology:BiologicalStructure
  is a sub-class of Class: simpleBioOntology:Structure
Class: simpleBioOntology:SecondaryStructure
  is a sub-class of Class: owl:Thing
  is a sub-class of Class: rdfs:Resource
  is a sub-class of Class: simpleBioOntology:BiologicalStructure
  is a sub-class of Class: simpleBioOntology:Structure
  is a super-class of Class: simpleBioOntology:AlphaHelix
  is a super-class of Class: simpleBioOntology:BetaSheet
Class: simpleBioOntology:DNA
  is a sub-class of Class: rdfs:Resource
  is a sub-class of Class: owl:Thing
  is a sub-class of Class: simpleBioOntology:Sequence
  is a sub-class of Class: simpleBioOntology:BiologicalStructure
  is a sub-class of Class: simpleBioOntology:Structure
Class: simpleBioOntology:SequenceCharacteristics
  is a sub-class of Class: rdfs:Resource
  is a sub-class of Class: owl:Thing
  is a super-class of Class: simpleBioOntology:SequenceLocation
  is a super-class of Class: simpleBioOntology:SequenceSimilarity
  is a super-class of Class: simpleBioOntology:SequenceLength
Class: simpleBioOntology:AlphaHelix
  is a sub-class of Class: rdfs:Resource
  is a sub-class of Class: owl:Thing
  is a sub-class of Class: simpleBioOntology:SecondaryStructure
  is a sub-class of Class: simpleBioOntology:BiologicalStructure
```

```
  is a sub-class of Class: simpleBioOntology:Structure
Class: xsd:short
  is a sub-class of Class: xsd:integer
  is a sub-class of Class: xsd:long
  is a sub-class of Class: xsd:int
  is a super-class of Class: xsd:unsignedShort
  is a super-class of Class: xsd:unsignedByte
  is a super-class of Class: xsd:byte
Class: xsd:int
  is a sub-class of Class: xsd:integer
  is a sub-class of Class: xsd:long
  is a super-class of Class: xsd:unsignedShort
  is a super-class of Class: xsd:unsignedByte
  is a super-class of Class: xsd:unsignedInt
  is a super-class of Class: xsd:byte
  is a super-class of Class: xsd:short
Class: xsd:long
  is a sub-class of Class: xsd:integer
  is a super-class of Class: xsd:unsignedShort
  is a super-class of Class: xsd:unsignedLong
  is a super-class of Class: xsd:unsignedByte
  is a super-class of Class: xsd:unsignedInt
  is a super-class of Class: xsd:byte
  is a super-class of Class: xsd:int
  is a super-class of Class: xsd:short
Class: xsd:byte
  is a sub-class of Class: xsd:integer
  is a sub-class of Class: xsd:long
  is a sub-class of Class: xsd:int
  is a sub-class of Class: xsd:short
  is a super-class of Class: xsd:unsignedByte
Class: xsd:unsignedInt
  is a sub-class of Class: xsd:nonNegativeInteger
  is a sub-class of Class: xsd:integer
  is a sub-class of Class: xsd:unsignedLong
  is a sub-class of Class: xsd:long
  is a sub-class of Class: xsd:int
  is a super-class of Class: xsd:unsignedShort
  is a super-class of Class: xsd:unsignedByte
Class: xsd:unsignedByte
  is a sub-class of Class: xsd:unsignedShort
  is a sub-class of Class: xsd:byte
  is a sub-class of Class: xsd:nonNegativeInteger
  is a sub-class of Class: xsd:integer
  is a sub-class of Class: xsd:unsignedLong
  is a sub-class of Class: xsd:long
  is a sub-class of Class: xsd:int
  is a sub-class of Class: xsd:unsignedInt
  is a sub-class of Class: xsd:short
Class: xsd:unsignedLong
  is a sub-class of Class: xsd:nonNegativeInteger
  is a sub-class of Class: xsd:integer
  is a sub-class of Class: xsd:long
  is a super-class of Class: xsd:unsignedShort
  is a super-class of Class: xsd:unsignedByte
  is a super-class of Class: xsd:unsignedInt
Class: xsd:nonNegativeInteger
  is a sub-class of Class: xsd:integer
  is a super-class of Class: xsd:unsignedShort
  is a super-class of Class: xsd:unsignedLong
  is a super-class of Class: xsd:unsignedByte
  is a super-class of Class: xsd:unsignedInt
Class: xsd:unsignedShort
  is a sub-class of Class: xsd:nonNegativeInteger
  is a sub-class of Class: xsd:integer
  is a sub-class of Class: xsd:unsignedLong
  is a sub-class of Class: xsd:long
  is a sub-class of Class: xsd:int
  is a sub-class of Class: xsd:unsignedInt
  is a sub-class of Class: xsd:short
  is a super-class of Class: xsd:unsignedByte
```

```
Class: rdf:List
  is a sub-class of Class: rdfs:Resource
Class: rdfs:Class
  is a sub-class of Class: rdfs:Resource
  is a super-class of Class: owl:Class
  is a super-class of Class: rdfs:Datatype
Class: rdf:Statement
  is a sub-class of Class: rdfs:Resource
Class: owl:Property
  is a sub-class of Class: rdfs:Resource
Class: xsd:string
  is a sub-class of Class: rdfs:Resource
Class: rdf:Property
  is a sub-class of Class: rdfs:Resource
  is a super-class of Class: owl:DatatypeProperty
  is a super-class of Class: owl:ObjectProperty
  is a super-class of Class: owl:FunctionalProperty
  is a super-class of Class: rdfs:ContainerMembershipProperty
  is a super-class of Class: owl:OntologyProperty
Class: owl:Ontology
  is a sub-class of Class: rdfs:Resource
Class: rdfs:Resource
  is a super-class of Class: simpleBioOntology:TertiaryZ
  is a super-class of Class: simpleBioOntology:Alignment
  is a super-class of Class: simpleBioOntology:MessengerRNA
  is a super-class of Class: simpleBioOntology:Gene
  is a super-class of Class: simpleBioOntology:Chromosome
  is a super-class of Class: simpleBioOntology:BetaSheet
  is a super-class of Class: simpleBioOntology:GlobalAlignment
  is a super-class of Class: rdf:List
  is a super-class of Class: owl:Thing
  is a super-class of Class: simpleBioOntology:DNA
  is a super-class of Class: simpleBioOntology:SequenceLocation
  is a super-class of Class: rdf:Statement
  is a super-class of Class: simpleBioOntology:Structure
  is a super-class of Class: rdfs:Class
  is a super-class of Class: owl:Nothing
  is a super-class of Class: simpleBioOntology:AminoAcidSequence
  is a super-class of Class: simpleBioOntology:Sequence
  is a super-class of Class: xsd:integer
  is a super-class of Class: simpleBioOntology:SecondaryStructure
  is a super-class of Class: simpleBioOntology:AlphaHelix
  is a super-class of Class: simpleBioOntology:Protein
  is a super-class of Class: owl:Ontology
  is a super-class of Class: rdf:Property
  is a super-class of Class: simpleBioOntology:TertiaryY
  is a super-class of Class: simpleBioOntology:TertiaryX
  is a super-class of Class: owl:Class
  is a super-class of Class: simpleBioOntology:ProteinName
  is a super-class of Class: xsd:string
  is a super-class of Class: simpleBioOntology:PrimaryStructure
  is a super-class of Class: rdfs:Literal
  is a super-class of Class: simpleBioOntology:Function
  is a super-class of Class: xsd:float
  is a super-class of Class: simpleBioOntology:SequenceCharacteristics
  is a super-class of Class: simpleBioOntology:ProteinFunction
  is a super-class of Class: simpleBioOntology:SequenceLength
  is a super-class of Class: owl:Property
  is a super-class of Class: simpleBioOntology:SequenceSimilarity
  is a super-class of Class: simpleBioOntology:LocalAlignment
  is a super-class of Class: simpleBioOntology:Polypeptide
  is a super-class of Class: simpleBioOntology:BiologicalStructure
  is a super-class of Class: simpleBioOntology:BiologicalFunction
  is a super-class of Class: owl:Restriction
  is a super-class of Class: simpleBioOntology:TertiaryStructure
Class: xsd:integer
  is a sub-class of Class: xsd:decimal
  is a sub-class of Class: rdfs:Resource
  is a super-class of Class: xsd:unsignedShort
  is a super-class of Class: xsd:nonNegativeInteger
  is a super-class of Class: xsd:unsignedLong
```

```
  is a super-class of Class: xsd:unsignedByte
  is a super-class of Class: xsd:unsignedInt
  is a super-class of Class: xsd:byte
  is a super-class of Class: xsd:long
  is a super-class of Class: xsd:int
  is a super-class of Class: xsd:short
Class: owl:Restriction
  is a sub-class of Class: owl:Class
  is a sub-class of Class: rdfs:Resource
Class: xsd:float
  is a sub-class of Class: rdfs:Resource
Class: owl:Class
  is a sub-class of Class: rdfs:Class
  is a sub-class of Class: rdfs:Resource
  is a super-class of Class: owl:Restriction
Class: rdfs:Literal
  is a sub-class of Class: rdfs:Resource
Class: xsd:date
Class: xsd:boolean
Class: xsd:decimal
  is a super-class of Class: xsd:integer
Class: xsd:duration
Class: xsd:nonPositiveInteger
Class: xsd:dateTime
Class: xsd:time
```