Searching Semantic Web Services: An Intelligent Agent Approach Using Semantic Enhancement of Client Input Term(s) and Matchmaker Algorithm

Duygu Çelik

Submitted to the
Institute of Graduate Studies and Research in partial fulfillment of
the requirements for the Degree of

Master of Science
in
Computer Engineering

Eastern Mediterranean University

September 2005, Gazimağusa

Approval of the Institute of Graduate Studies and Research

Prof. Dr. Ayhan Bilsel
Director

I certify that this thesis satisfies the requirements as a thesis for the degree of Master of Science in Computer Engineering

Assoc. Prof. Dr. Hasan Kömürcügil
Chair, Department of Computer Engineering

We certify that we have read this thesis and that in our opinion it is fully adequate in scope and quality as a thesis for the degree of Master of Science in Computer Engineering.

Assoc. Prof. Dr. Atilla Elçi
Supervisor

Examining Committee

1. Assoc. Prof. Dr. Atilla Elçi
2. Prof. Dr. Erden Başar
3. Assoc. Prof. Dr. Zeki Bayram
4. Asst. Prof. Dr. Mustafa Rıza

ABSTRACT

Intelligent Semantic Web Service Finder

This thesis proposes a simple mechanism to discover semantic web services satisfying client requirements. The increase in web services and lack of semantic base in search mechanisms of UDDI make it difficult for clients to find a required web service. It is proposed to develop an intelligent agent to discover required web services by a client from the web. The system uses OWL-S for describing semantics of web services and discovers appropriate semantic web services through these semantic descriptions. OWL-S allows semantic description of web services and hence, the intelligent agent is able to understand predefined semantic descriptions of semantic web services, that extracts necessary profile information from those web services and decide appropriate service(s) based on the requirements of a client. The intelligent agent interacts with the client and Semantic Web Services.

The proposed system further combines aspects of two research topics (Smart Web Query Engine and Matchmaking Algorithm of OWL-S/UDDI Matchmaker) to facilitate the finding of semantic web services for a client. First, the intelligent agent takes clients request word(s) and enhances the client request term(s) with its *synonym* terms and *is_a* related terms. Then all related terms are sent to UDDI for the retrieval of related web services address and information. Afterwards the agent connects to all of the related web services' OWL-S files and extracts the necessary input/output information about them. The intelligent agent evaluates the input/output information of the client and input/output information of SWS (profile

information in semantic description of SWS) according to the matchmaking algorithm.

The proposed system is implemented in the JAVA platform by using NETBEANS 4.0. Also, the Financial Investments, Vehicle, Software and Adhesive ontologies are created in Protégé 3.1. Some semantic web services are created for our test applications in Protégé 3.1 as well. The client sends some request parameters to the system such as the term "Bond" and the system takes the term. The system checks the term "Bond" in our predefined ontologies. Then the system sends to the client all retrieved ontology names that include the term "Bond". The client selects the most relevant ontology name according to his/her search criteria. The system works only on selected ontology from then on. The system accesses the selected ontology and extracts all related terms such as *synonym* terms and is_a related terms according to the term(s) entered by the client. Then all terms are sent to the UDDI and the system receives all semantic web services information and their addresses. The system asks for input/output parameters from client after that connects to all those returned web services and extracts profile information by accessing their OWL-S file(s) or their semantic descriptions. Finally, the system starts to perform the matchmaking algorithm between all input/output parameters of client and input/output parameters of returned web services. The system is able to specify some relationships between those input/output parameters such as EXACT, PLUGIN, SUBSUME and FAIL. According to those relationships the system serves appropriate web services to the client based on client request.

ÖZET

Akıllı Anlamsal Ağ Servisi Arayıcısı

Bu tez çalışması bir kullanıcının isteğine bağlı olarak anlamsal ağ servislerinin keşfedilmesini sağlayan akıllı bir mekanizmayı önermektedir. Ağ servislerinin sayının artışı ve anlamsal bir arama yapılamaması yüzünden kullanıcının isteğine uygun ağ servislerinin UDDI'dan elde edilmesi oldukça zor olmaktadır. Bu çalışma bunun için bir akıllı ajan geliştirmeye ve istenilen ağ servislerini ağ üzerinden akıllı ajan tarafından kolayca elde etmeyi amaçlamaktadır. Bu sistem anlamsal ağ servislerini oluşturmak için kullanılan OWL-S dilinden yapılanarak ve istenilen ağ servisi bu anlamsal tanımlar arasında arayarak uygun ağ servislerini kullanıcıya kolaylıkla sunmaktadır. Akıllı Ajan daha önceden anlamsal olarak oluşturulmuş anlamsal ağ servis'lerinin içindeki anlam bilgilerine girerek ve gerekli bilgiyi oradan ayrıştırarak kullanıcının isteğine uygun bir ağ servis olup olmadığı hakkında karar verebilmektedir. Bu akıllı ajan hem kullanıcı ve hem de anlamsal ağ servisleriyle sürekli iletişim kurabilmektedir.

Bu sistem iki farklı araştırma konusunun birleşimini kullanmayı tasarlamaktadır: Akıllı Ağ Sorgu Motoru ve Uyum Algoritması. Bu iki araştırma konusunu kapsayarak kullanıcıya ağ servislerini bulmasını kolaylaştıran akıllı bir sistem içermeyi hedeflemektedir. Öncelikle, akıllı ajan kullanıcının istek kelimelerini almaktadır ve bu kelimeleri onun *eşanlamlı* ve *bir_örneğidir* bağlantılı terimlerle çoğaltmaktadır. Daha sonra tüm ilgili bulunan terimler UDDI' YA tüm ilgili ağ servislerinin adreslerini ve bilgilerini elde edebilmek için gönderilmektedir. Daha sonra ajan tüm bu ilgili ağ servislerin OWL-S dosyalarına bağlanarak tüm gerekli girdi/çıktı değerlerini ayrıştırmaktadır. Akıllı ajan,

kullanıcının girdi/çıktı değerleri ile her ağ servisin ayrıştırdığı girdi/çıktı değerlerini (SWS'lerdeki anlamsal tanımlamalarında bulunan profil bilgileri) uyum algoritmasına göre değerlendirmektedir.

Önerilen sistem NETBEANS 4.0 kullanılarak JAVA platformuna tamamlandı. Ayrıca bazı gerekli ontolojiler hazırlandı örneğin Finasal Yatırımlar, Araçlar, Yazılım ve Yapıştırıcılar ontolojileri Protege 3.1 kullanılarak yartatilmistir. Yine Protege 3.1 kullanılarak bazı anlamsal ağ servisleri bizim test uygulamalarımız için yaratılmıştır. Kullanıcı sisteme bazı istek parametreleri göndermektedir örneğin "Bono" terimi ve sistem bu terimi almaktadır. Bu sistem önceden hazırladığımız ontolojilerimizin içine girerek hangi ontolojilerin bu terimi içerdiğini kontrol etmektedir. Daha sonra sistem bulduğu tüm ontolojilerin isimlerini kullanicaya göndermektedir. Kullanıcı en ilgili ontolojinin adini seçmektedir. Bundan sonra sistem sadece bu ontoloji üzerinde çalışmaktadır. Bu sistem seçilen ontolojiye giriş yapmakta ve tüm ilgili kelimeleri kullanıcının girdiği terim veya terimlere göre ayrıştırmaktadır örneğin; *eşanlamlı* ve *bir_örneğidir* bağlantılı tüm terimler. Daha sonra tüm bulunan ilgili terimler UDDI'ya gönderilmektedir ve sistem tüm ilgili anlamsal ağ servislerinin adres ve bilgilerini almaktadır. Ayrıca sistem kullanıcıdan girdi/çıktı parametrelerini istemektedir daha sonra sistem tüm ilgili ağ servislerine bağlanmaktadır ve onların anlamsal tanımlamalarına veya OWL-S dosyalarına girip profil bilgilerini ayrıştırmaktadır. Tüm bunlardan sonra, sistem kullanıcının girdi/çıktı parametreleri ve ağ servislerinin girdi/çıktı parametreleri arasında uyum algoritmasını çalıştırmaya başlamaktadır. Bu sistem bu parama terler arasında bazı ilişkiler belirleyebilmektedir örneğin; KATI, KAPSANMA, KAPSAR ve BASARISIZ. Bu ilişkilere göre sistem uygun ağ servislerini kullanıcının isteğine bağlı kalarak kullanıcıya sunmaktadır.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

SWS: Semantic Web Services

IA: Intelligent Agent

UDDI: Universal Description, Discovery and Integration

WSDL: Web Services Description Language

SOAP: Simple Object Access Protocol

OWL: Web Ontology Language

OWL-S: Web Ontology Language for Services

CHAPTER I

INTRODUCTION

Currently, web services interact by passing XML data, with data types specified using XML Schema. Simple Object Access Protocol [SOAP] can be used as the communication protocol [1], and the I/O signatures for web services are given by Web Services Description Language [WSDL] [2]. UDDI stands for Universal Description, Discovery and Integration [3] and provides the means to publish and discover web services through a UDDI registry.

The topic of this thesis is a discussion of the development of an Intelligent Agent (IA) that understands ontologies, concepts of Semantic Web Services and connects to each of these services. The IA extracts necessary information choosing Semantic Web Services (SWS) that are appropriate for the client based on client's request. The proposed IA is used to service the client by looking at the semantic description of the SWS. It presents semantically searched results to the client detailing the services selected based on the request received by IA. First, the agent enhances the client input term with *synonym* or some related words such as *is_a*, then it checks if there is a match between client's request and the profile information (ontologies) of SWS.

In this thesis, ontology languages are used to describe formal specification of web services to be able to retrieve semantics, and to provide the means of defining complicated class properties. A new semantic technology OWL-S, available on Semantic Web Central site [4], can be used to develop SWS. This is a rapidly growing research area. OWL-S and its Editor are preferred since they are new, effective methods, and are suitable for IA functioning of our system.

There are some difficulties in discovering required web services by a client due to increase of web services and lack of semantic parts in current web service technologies such as UDDI and WSDL. Due to this fact, the web requires an intelligent system which can be handled by using Semantic Web. The lack of semantic parts in web services technologies and also increasing number of web services are making it too difficult for finding appropriate web services according to the client's request.

The proposed system helps to find appropriate SWS that provide service to clients. The proposed system includes an intelligent agent placed between client and SWS. The intelligent agent provides communication between clients and SWS; therefore, the intelligent agent should be able to understand the semantics of SWS and communicate the client's semantic information to SWS and vice versa. The agent should decide and suggest if a particular service is useful or not for all services, according to a client's request, as an Intelligent Semantic Web Service Finder (ISWSF) for clients. Such an agent is called ISWSF agent. Based on the aforementioned scenario we classify the intelligent agent work in two main parts: Semantic Enhancement of Input Terms and Matchmaking Algorithm.

The thesis is arranged as follows: section 2 discusses SWSs and OWL-S and provides a survey on smart search engines and UDDI search mechanisms. Section 3 details the architecture of the system developed during this research. The architecture focuses on two main functions of our proposed intelligent agent (Semantic Enhancement of Input Terms and Matchmaker Algorithm). Section 4 presents the implementation of the system and then Section 5 is showing two case studies of the system. Finally, the last part is the conclusion to this thesis.

CHAPTER II

SEMANTIC WEB SERVICES AND OWL-S

The Semantic Web allows users to locate, select, employ, compose, and monitor Semantic-based web services automatically. Web sites should be able to use a standard ontology, consisting of a set of basic classes and properties, for declaring and describing services, the ontology structuring mechanisms of OWL provides an appropriate, Web-compatible representation language framework for the realization of these requirements.

We need SWS, because, SWS provide machine-readable descriptions of web services, which enable automated discovery, negotiation with, composition, execution, and monitoring of web services. Furthermore, ontology languages provide a way for describing their formal specification and have the ability to define properties of the Web Services. Through property description capabilities of ontology languages, descriptions of services can be defined more accurately, and services can be related to other services or resources more easily.

Well accepted standards WSDL and SOAP make it possible to dynamically invoke Web services. That is, when the semantic of the service to be used is known, its WSDL description can be accessed by a program such as an interface, which uses the information in the WSDL description to provide dynamic access to service.

Therefore, to be able to use the Web services to their full potential, their semantic information should be available. In this way, not only humans can query the service registries to discover and compose services, but more importantly, this also opens up the way for software agents to automatically exploit Web services on behalf of users since formal semantic descriptions are machine processable.

This thesis attempts to motivate OWL-S for discover Semantic Web Services. OWL-S is characterized by three modules that are Service Profile, Process Model and Grounding. The service *profile* describes the capabilities of web services for advertising and discovering services; the *process model*, which gives a detailed description of a service's operation; and the *grounding*, which provides details on how to interoperate with a service, via messages.

Furthermore, OWL-S Editor is a tool that allows easy, intuitive OWL-S service development for developers and provides a variety of special-purpose capabilities to facilitate SWS design. The editor is implemented as a plug-in to the Protégé OWL ontology editor. As a result, the Semantic Web Services are defined semantically, because using Semantic Web Services description of service properties becomes straightforward. Furthermore, using Semantic Web Services enables the IA to search for services based on their properties.

Thus, our intelligent agent can extract needed I/O information from properties of the SWS easily. Those SWSs are returned by UDDI and they found relative with client entered keywords. Success of find of appropriate web services is depending on matching for client input/output and the input/output of all returned web sevices from UDDI.

As mentioned before, the agent has two main tasks and it executes them sequentially. The first task is Semantic Enhancement of Input Terms whereas the second is Matchmaking Algorithm. Finally, this chapter is mentioned about our purposed system and next sections investigate some researches that are applied on those IA tasks.

2.1. Smart Web Query Engine

Smart Web Query Engine [5] discusses search engines. Search engines are very important tools for the people to get information from Internet, but the low-accuracy and low-recall exists widely in current search engines.

Search Engines provide only primitive data query capabilities and require a detailed syntactic specification to retrieve relevant data. Also, web data can include PDF documents, images and sound clips that are difficult to be queried. Smart Web Query Engine project [5] investigates a new web query method which is called Smart Web Query (SWQ) method for semantic retrieval of web data. The SWQ method uses domain semantics represented as context ontologies. These context ontologies are specified and formulate appropriate web queries to search.

SWQ architecture can be divided into three parts. First part is Internet users and their user interfaces, second part is SWQ engine and its components and the third part is some search engines. The second part is the main part of SWQ engine and includes two components. These components are Context Ontologies and Semantic Search Filters. Context Ontologies are using for application domains and search filters are used to improve search precision based on properties in the context ontology.

The user interface of SWQ is similar to traditional search engines. A user gives some set of keywords that are separated using some Boolean operators and some search parameters such as maximum number of pages to consider for search operation. The user enters some keywords and search parameters; processing this input the engine builds a parse tree for the users' query. User keyword is enhanced with "*synonym*", "antonym" and some related terms such as "is a" or "acts on" and the engine prepares an appropriate query which is sent to some classical search engine(s) to retrieve relevant pages. Next, the engine applies some semantic search filters to returned relevant pages.

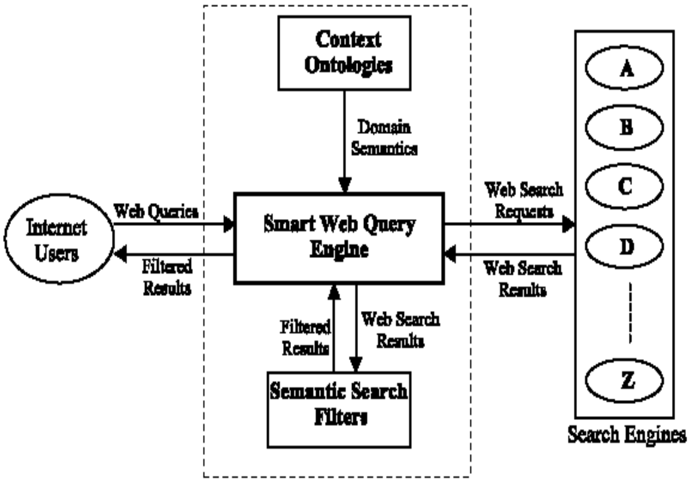

Figure1. The System Architecture of Smart Web Query Engine [5].

This thesis found this approach to be helpful in the task and hence uses some concepts of SWQ to retrieve relevant web services from UDDI. The first task of the intelligent agent, proposed in this thesis, performs SWQ logic for Semantic Enhancement of Input Terms.

2.2. UDDI Search Mechanism for Web Services

UDDI is a standard for discovery of Web Services that stands for Universal Description, Discovery and Integration. The UDDI Project is an industry plan that is working to enable businesses to quickly, easily, and dynamically find and carry out transactions between each other. Services can be searched by name, location, business, binding or by TModels in UDDI.

The UDDI registry can be divided in to two categories namely UDDI-Web Services and UDDI-Business Registry. The Web Service developers publish their web services to the UDDI registry. Once published, the UDDI registry holds information of web service description and address. The UDDI allows clients to search this registry, find the planned service and retrieve its details. These details include the service invocation point as well as other information to help identify the service and its functionality.

Nowadays, many researchers are targeting on adding semantics to UDDI for the purpose of enhancing the discovery mechanism of UDDI. One such study, developed by Paolucci et al, is called OWL-S/UDDI Matchmaker [6, 7, 8 and 9] that describes an algorithm for matching service advertisement and service request. Paolucci et al developed a Matching Engine that performs flexible matches that recognize the degree of an advertisement in UDDI and a client request. Detailed information about this subject in available in next sub section of this thesis.

UDDI has some disadvantages for discovering web services. One of them is that UDDI uses XML to describe its data model. UDDI guarantees syntactic interoperability, but it fails to provide a semantic description of its content. Paolucci et al adopted OWL-S that can be used to describe the capabilities of web services. They implemented a matchmaker engine in UDDI. The engine performs matchmaking algorithm to find appropriate web services according to client request.

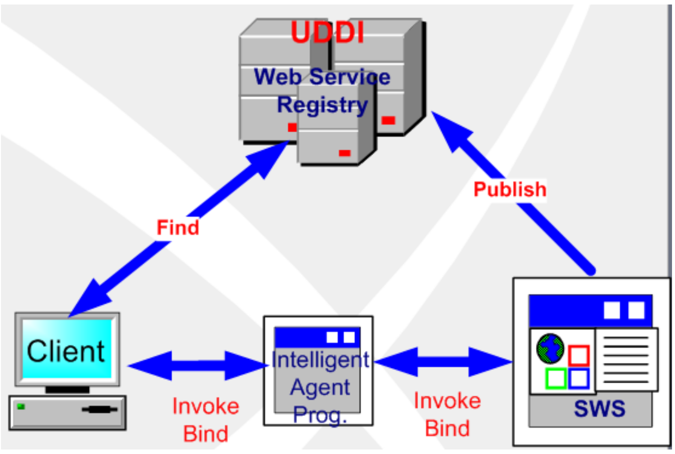

Figure 2. The General Architecture of ISWSF.

In this study, UDDI discovery part will be performed classically; yet we proposed to place an intelligent agent between client and SWSs instead of changing the structure of UDDI standard (See Figure 2). Next section details the matchmaking algorithm.

2.2.1 Matchmaking Algorithm

Matchmaking Algorithm is the algorithm implemented in UDDI by Paolucci et al [6, 7, 8 and 9]. UDDI records advertisements of services and supports search of services that provide a set of requested functionalities. But, in its current form UDDI, does not provide any support for finding services on the basis of what tasks they perform. Advertisement and client request refer to OWL concepts and the associated semantic. The system tries to recognize semantic matches despite their syntactic differences and difference in modeling abstraction between all advertisements in related advertisements and a client request. No matching occurs if a request does not recognize any advertisement. A ranking Function which distinguishes multiple degrees of matching is provided in [6].

```
match(request) {
   recordMatch= empty list
   forall adv in advertisements do {
      if match(request, adv) then
          recordMatch.append(request, adv) }
   return sort(recordMatch);}
```

Figure 3. Main Control Loop [6].

The above figure shows main control loop of the algorithm. It is matching requests and all advertisements stored by the registry. Whenever a match between a request and an advertisement is found then it is recorded and scored. This process is provided to find the matches with the highest degree.

```
outputMatch(outputsRequest, outputsAdvertisement) {
   globalDegreeMatch= Exact
   forall outR in outputsRequest do {
      find outA in outputsAdvertisement such that
         degreeMatch= maxDegreeMatch(outR,outA)
         if (degreeMatch=fail) return fail
         if (degreeMatch<globalDegreeMatch)
            globalDegreeMatch= degreeMatch
   return sort(recordMatch);}
```

Figure 4. Algorithm for Output Matching [6].

A match is found if and only if for each output of the request, there is matching output in the advertisement. Degree of success depends on the degree of match detected. If one of request’s output is not matched by any of advertisement's output then the match fails. The matching events have same structure for inputs. (Figure 4)

```
degreeOfMatch(outR,outA):
   if outA=outR then return exact
   if outR subclassOf outA then return exact
   if outA subsumes outR then return plugIn
   if outR subsumes outA then return subsumes
   otherwise fail
```

Figure 5. Rules for Degree of Match Assignment [6].

Figure 5 shows the rules, these rules are separated as **exact, plug in, subsume** and **fail. OutR** symbolizes one output of the request whereas **OutA** symbolizes output of the advertisement.

Exact: If OutR and OutA are same or if OutR is an immediate subclass of OutA. For example, given the ontology fragment like Figure 6, the degree of match between a request whose output is Sedan and an advertisement whose output is Car is exact.

Plug in: If OutA subsumes OutR, then OutA is assumed to cover OutR or in other words OutA can be plugged instead of OutR.

Subsume: If OutR subsumes OutA, then the provider may or may not completely satisfy the requester. Hence this match is inferior than the plug in match.

Fail: A match is a fail if there is no subsumption relation between OutA and OutR.

There is a determination of priority level between those rules.

- The **exact** is best level degree.
- The **plug in** matches is next best level degree. Because the returned output can probably be used instead of what the requester expect.
- The **subsume** is third best degree because, requirements of the requester are only partially satisfied. Namely, the advertised service can provide only some specific cases of what requester desires.
- The **fail** is lower level degree and it means an acceptable result.

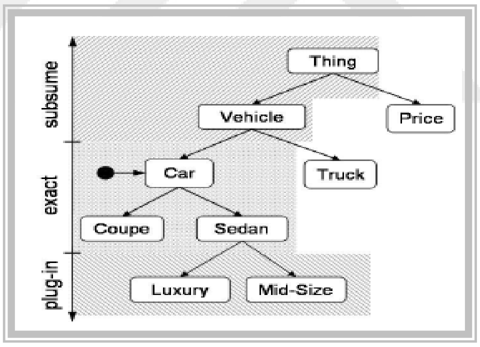

Figure 6. Advertisement Propagation of Vehicle Ontology [6].

Finally, the thesis purpose is implement matchmaking algorithm on the intelligent agent instead of UDDI. UDDI will work classically in our study. The agent will perform the matching algorithm between the client request and SWSs input/outputs information. If the agent will find any similarity according to above rules then it will decide that web service useful for the client or not useful. After that all elimination of the returned web services by the UDDI, the agent will prepare an interface and send it to client as a result.

CHAPTER III

INTELLIGENT SEMANTIC WEB SERVICE FINDER (ISWSF) SYSTEM ARCHITECTURE

3.1. System Architecture of ISWSF

The figure 7 is showing system architecture of the ISWSF. The system architecture includes ISWSF system (our proposed Intelligent Agent), a Client who tries to find appropriate web services, an Ontology Database that defines relations between terms, UDDI registry and returned related web services by UDDI.

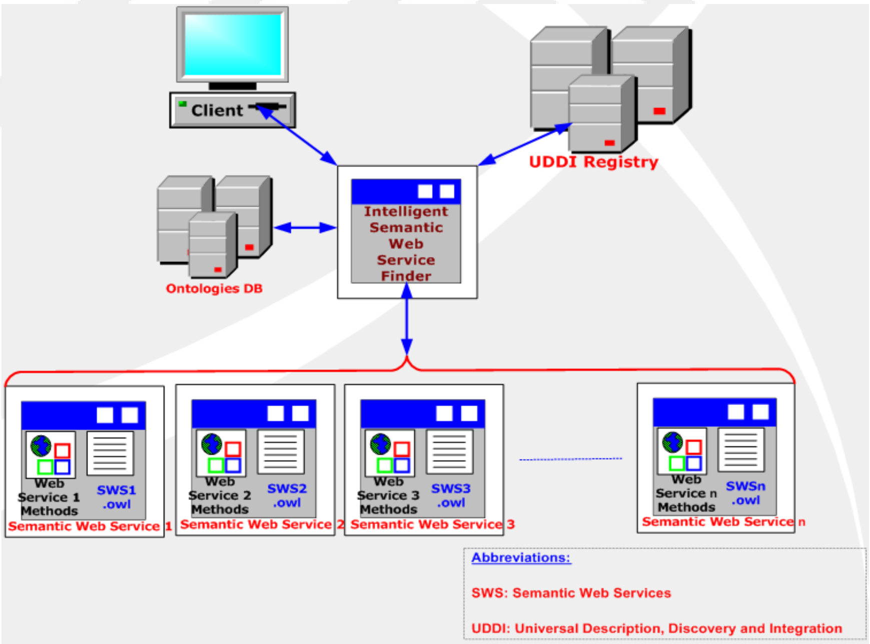

Figure 7. The System Architecture of ISWSF.

The purpose of this system is to guide a client looking for an appropriate web service. This is achieved by using an intelligent agent through SWS's semantic descriptions according to client's request. This request is understood by an intelligent agent in an appropriate manner and the agent tries to find suitable SWS through their semantic descriptions (ontologies/concepts).Thus, clients are able to find required web services in an efficient way and with ease in matching their requirements. The agent interacts with the client and SWS from the beginning of the process till the end.

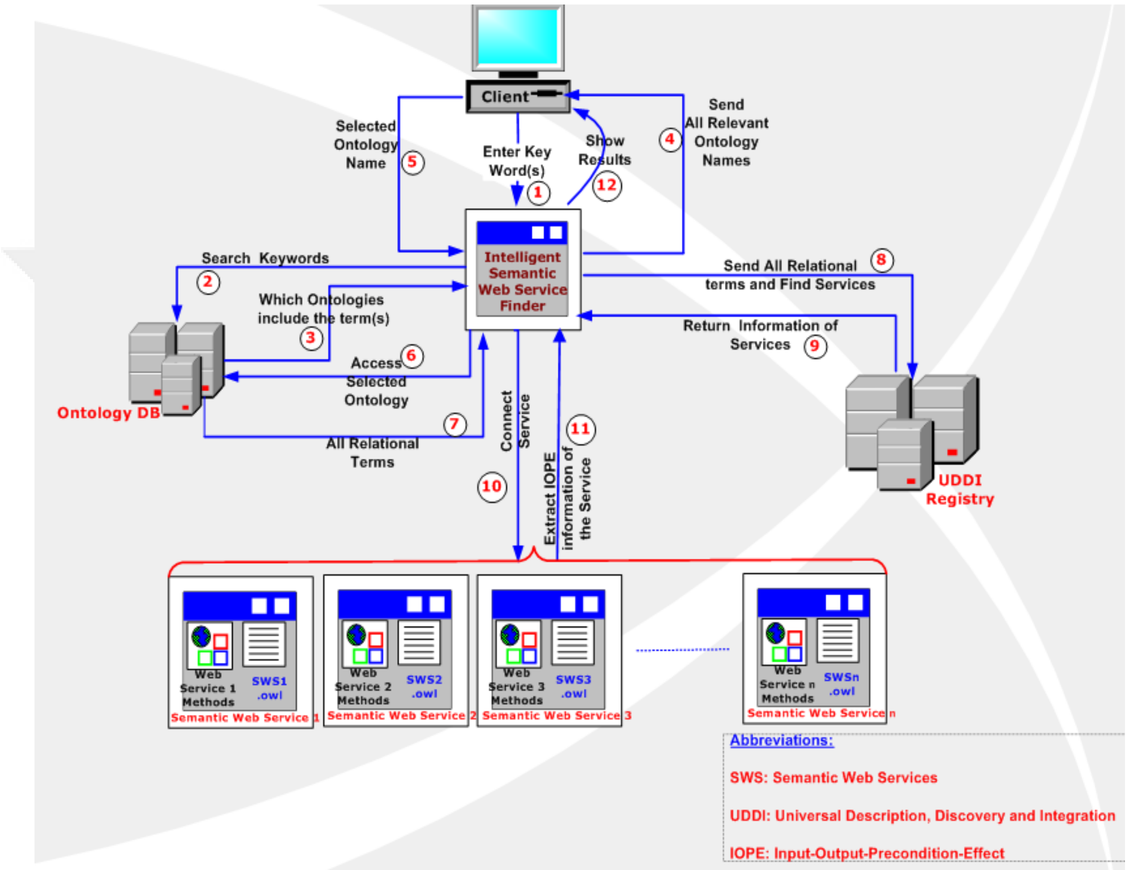

Figure 8. The Proccess Chart of ISWSF.

Web Service technologies such as UDDI and Semantic Web technologies such as OWL-S or OWL are used to develop this project. OWL-S is used to add semantics into web services and IA is able to understand meaning of semantic contents enabling it to use the full potential of SWS. The response generated by an IA will then provide references of the selected services to the clients as a result. Semantic information about services such as input, output, and precondition is stored in the profile of the services.

Figure 8 shows the process chart of the ISWSF system. Firstly, client enters request word(s) as an input to the system (ref. 1 in Figure 8). Then, the IA takes entered word from client. Next, the agent checks all ontologies against this entered word (ref. 2-3 in Figure 8); then prepares an interface to ask the client which ontology is related with its search (ref. 4 in Figure 8).

The client selects the relative ontology name and sends it to agent (ref. 5 in Figure 8). The agent accesses the selected ontology and then looks for *synonym*/*is_a* words which are stored in selected ontology by the client and enhances (ref. 6-7 in Figure 8). All related terms are the entered word with *synonyms* and *is_a* words. In this project, some ontologies were prepared for defining relations between client's terms and ontology's terms. The IA is able to find most relevant ontology according to the client request and then operates on that ontology to extract the terms similar to the term entered by the client.

All operations at this point are enhanced by the agent using the client input term and *synonym* terms or *is_a* terms. For example the term "Bond" has two *synonym* terms that are "Stock" and "Share". It also has one *is_a* relation that is "Bond is an Instrument". All those operations operate similar to SWQ Engine project developed by Storey et al [5].

At this point the system finished one of its main tasks. It is called Semantic Enhancement of Input Term(s) and which is highlighted in Figure 9. The Semantic Enhancement of Input Terms includes three main components of the system which are Client, Ontology Database and the ISWSF.

After the process is complete then the system will perform the matchmaking step. This step is second main task of the proposed system.

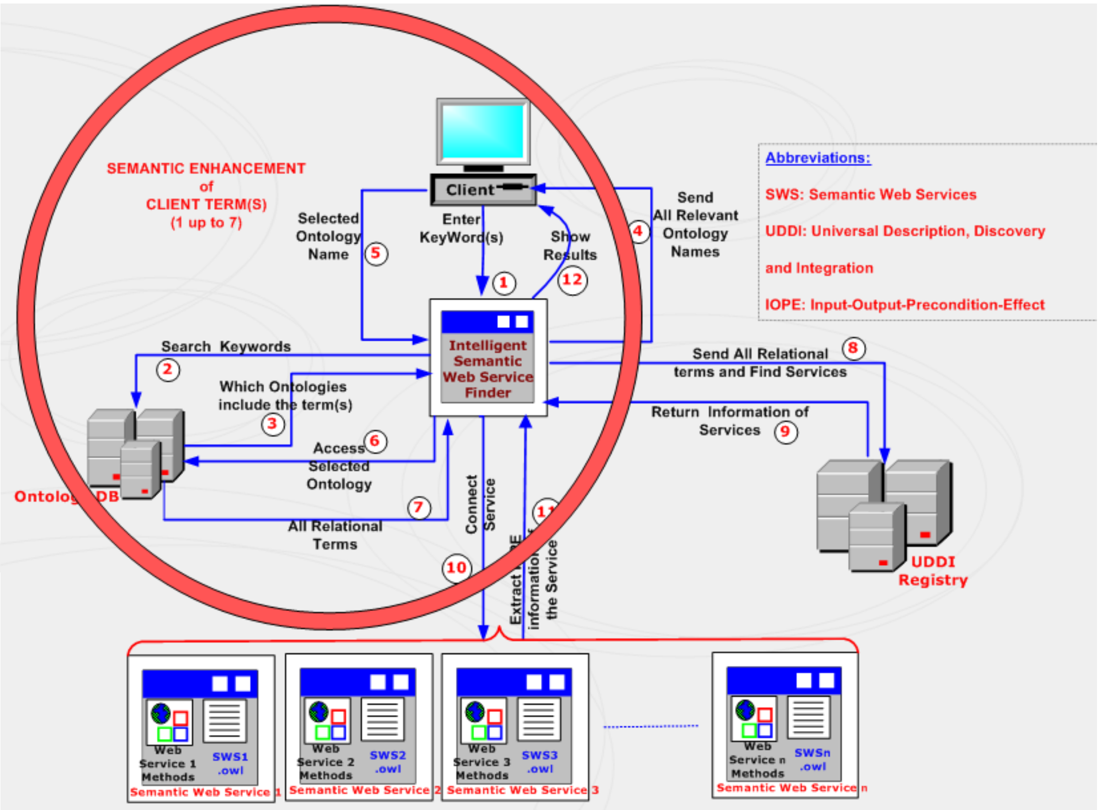

Figure 9. Semantic Enhancement of Input Term(s).

All found related terms are sent to the UDDI through a request to retrieve relevant web services of UDDI (ref. 8-9 in Figure 8). For example, if we send the "bond" term to UDDI directly, the UDDI may send back irrelevant services or is unable to find any services recoded as "bond" in UDDI. So, the system should first find terms related to the term "bond" and then send the service request to UDDI. Defining relations between terms is available in ontologies. Thus, the project uses Protégé 3.1 for creation of ontologies, its instances and relations. Protégé 3.1, developed by the Stanford University [10], is a very useful tool for creation of ontologies.

Remaining operations belonging to the operations of the agent are similar to that of matchmaker engine project. The UDDI returns related service names and address information sent to agent through client request (depicted as 8-9 see Figure 8). Now, the agent connects to those services and extracts I/O information of the services' profile information based on their OWL-S ontologies (depicted as 10-11 see Figure 8).

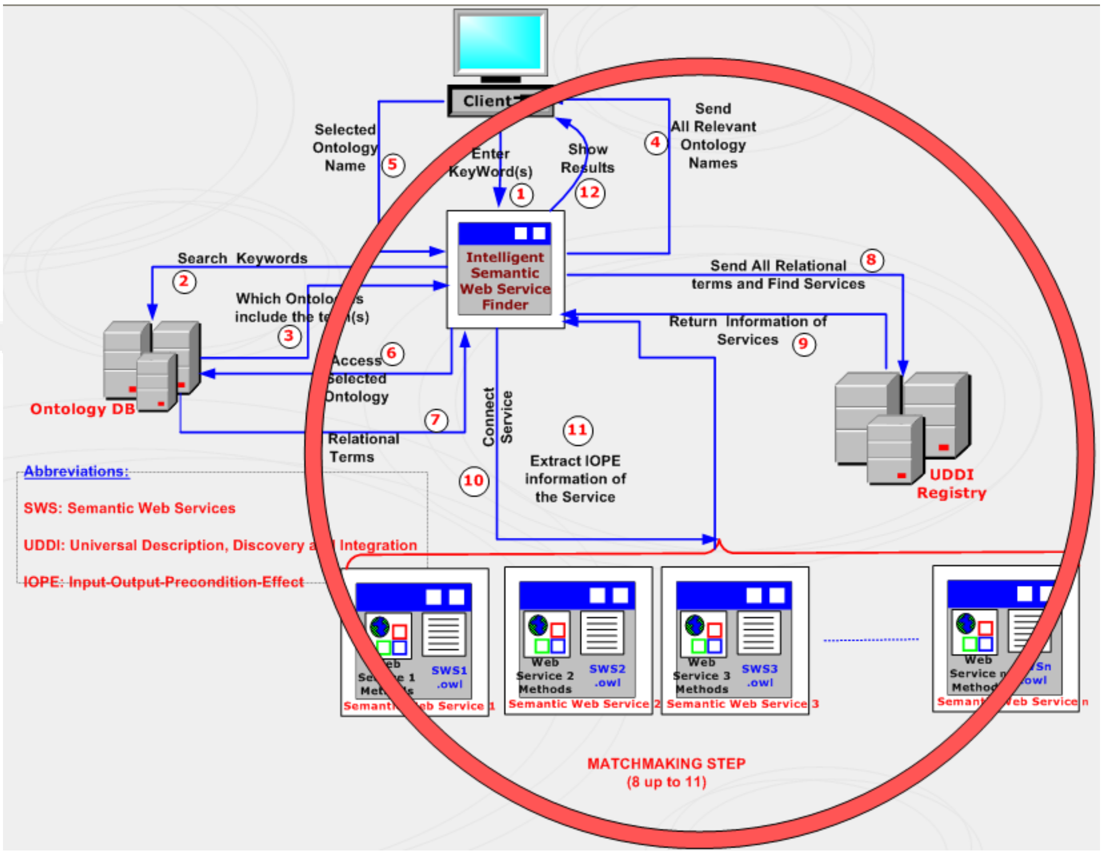

Figure 10. Matchmaking Step.

The client input information and output information will be compared with those services' input and output data. It was necessary to have an efficient algorithm here and matchmaker algorithm was found to be best suited for this purpose [6].

After matchmaking step, the agent will prepare an interface including eliminated SWS names and addresses than sends it to client as an output at step 12 see Figure 8.

At this point the system finished last one of its main tasks. It is called Matchmaking Step and which is highlighted in Figure 10. The Matchmaking Step includes three main components of the system which are semantic descriptions of returned SWS's (OWL-S files), UDDI Registry and the ISWSF.

As mentioned above, there are two main roles in intelligent agent's tasks list. This section gave brief description of the system architecture and the tasks of the main procedures in the system.

Next section details information on how the intelligent agent is implemented in JAVA and how the IA extracts the necessary input and outputs information from the returned semantic web services from UDDI. Also, next section will give detailed information on how Semantic Enhancement of Client Input Terms and Matchmaking Step are implemented in the proposed system.

CHAPTER IV

IMPLEMENTATION

The ISWSF system is developed in Java platform. The system used some important and useful tools that are developed for semantic web technologies such as OWL, OWL-S. Those tools were helpful for retrieving necessary information from OWL and OWL-S files. Next section is mentioning those tools and the system implementation. Those tools are imported in Java Net beans 4.0 which are Jena 2.0 OWL API and OWL-S API. The another sub section is mentioning about Protégé 3.1 (Section 4.2) which is used for creating SWSs and Ontology files such as financial investments ontology (see Appendix K). Next sections are giving detail information of developed classes for performing the IA tasks (Sections 4.3 to 4.6).

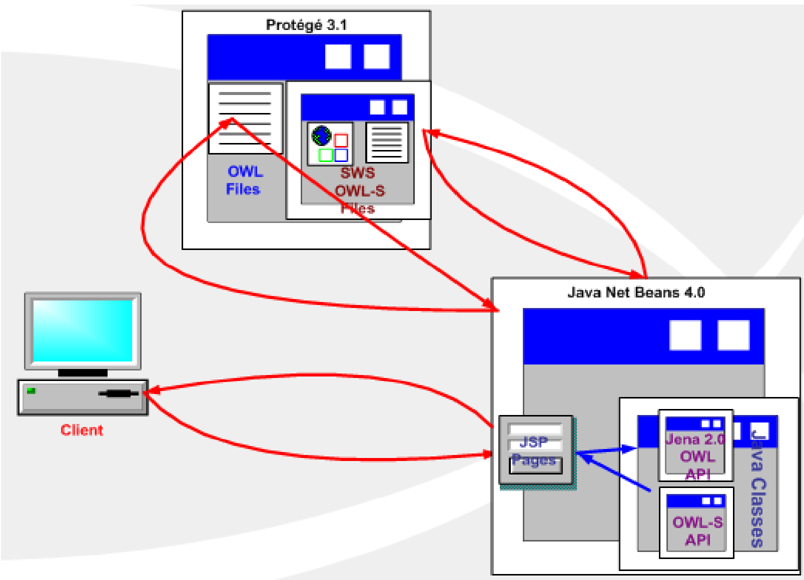

Figure 11. Used modules for implementation of ISWSF.

4.1. Java Net Beans 4.0

The system is implemented in Java Net Beans 4.0. The Intelligent Agent for our proposed system is developed by using this tool, because, Java Net Beans 4.0 allows our import necessary packages into the IA project. These packages were some necessary APIs for OWL or OWL-S. To use those packages Net Beans has to know where to find them. The process is adding some JAR files for those APIs. Some necessary APIs and used tools are listed below with short descriptions.

4.1.1 Jena 2.0 Ontology API

Jena is a Java framework for writing Semantic Web applications. The Jena2 ontology API is intended to support programmers who are working with ontology data based on RDF. Specifically, this means support for OWL, DAML+OIL and RDFS. A set of Java abstractions extend the generic RDF Resource and Property classes to model more directly the class and property expressions found in ontologies using the above languages, and the relationships between these classes and properties [11]. IA project used Jena2 ontology API for extracting some necessary information from an OWL file and then used the extracted information to make its own tasks. For example, the IA first open the one of all owl files and then extract all terms and then compare them with all client terms one by one.

```
InputStream in = FileManager.get().open( ontFile );
if (in == null) {
throw new IllegalArgumentException( "File: " + ontFile + " not found");
 }

OntModel loaded_model = ModelFactory.createOntologyModel();
loaded_model.read(new InputStreamReader(in), "");
Iterator it =   loaded_model.listIndividuals();

while (it.hasNext())
{
Individual individual = (Individual) it.next();
String term=individual.getLocalName().toString();
for(int j=0; j<ClientKWL.length; j++){
```

```
if(term.equalsIgnoreCase(ClientKWL[j]))
 {
OntNameList.add(individual.getRDFType().getLocalName());
                }//if
          } //for
     }//while
```

Figure 12. The IA Used Jena API for Retrieve a Relation Information between Client Term and its Domain Ontology File

4.1.2 OWL-S API

OWL-S API provides a Java API for programmatic access to read, execute and write OWL-S (formerly known as DAML-S) service descriptions. The API supports to read different versions of OWL-S (OWL-S 1.0, OWL-S 0.9, DAML-S 0.7) descriptions. The API provides an Execution Engine that can invoke Atomic Processes that has WSDL groundings, and Composite Processes that uses control constructs Sequence, Unordered, and Split [12].

IA project used OWL-S API for extracting some necessary Service Profile information from a returned SWS OWL-S file by the UDDI and then used the information to perform matchmaking step. For example, below figure is showing the IA first open the one of all owl files and then extract all inputs and outputs terms and then compare them with all client inputs and outputs terms one by one.

```
IA_Match o=new IA_Match(new java.io.File(SWSFiles[k]).toURI().toString());
InputList inputList=profile.getInputs();
showinp.add(step+".Semantic Web Service");
step=step+1;
for(int j=0; j<ClientInp.length; j++) {
for(int i=0;i<inputList.size();i++) {
String relinp=getRelation(ClientInp[j],inputList.inputAt(i).getLocalName());
```

Figure 13. The IA Used OWL-S API for Retrieve Relation Information between Client I/O Terms and Each SWS I/O Terms.

4.1.3 Java Server Pages (JSP)

Java Server Pages (JSP) technology enables Web developers to rapidly develop and easily maintain, information-rich, dynamic Web pages that leverage existing business systems. As part of the Java technology family, JSP technology enables rapid development of Web-based applications that are platform independent. JSP technology separates the user interface from content generation, enabling designers to change the overall page layout without altering the underlying dynamic content [13].

The system is required to use many JSP pages because the JSP is separating the page logic from its design and display and supporting a reusable component-based design. The JSP technology makes it faster and easier. In this thesis many JSP files are developed and next sections will discuss each one with briefly.

4.2. Protégé 3.1

Protégé is a free-open source ontology editor and knowledge-base framework. Protégé is based on Java, is extensible, and provides a foundation for customized knowledge-based applications. Protégé supports Frames, XML Schema, RDF(s) and OWL. It provides a plug-and-play environment that makes it a flexible base for rapid prototyping and application development [14].

4.2.1 Creating Related Ontologies in Protégé 3.1

IA project used Protégé 3.1 like a classical database system, separately defined classes of information (schema) and stored instances of these classes; Protégé 3.1 makes it easier to work simultaneously with both classes and instances. The Protégé 3.1 used for creating ontologies and its classes, individuals and properties. The IA is able to use those ontologies for retrieve relations between those classes and its individuals in the system. For example, Financial_Investment ontology created by using this tool (see Appendix K). There are many classes, individuals and data type properties in this ontology. Those properties are providing to make relation between classes and its individuals.

Below figure is showing a small part of the Financial_Investment ontology and IA used this owl file to retrieve relations between client terms and its "*synonym*" or "*is_a*" related terms which are predefined in the Financial_Investment ontology. The owl code is including the term "Bond" which has two *synonym* terms that are "Share" and "Stock". Also, the term "Bond" has one "*is_a*" relation which is "Instrument" term. When the client enter the term "Bond" then the IA will check all existing ontologies which one(s) are having the term "Bond". The IA extracts all *synonym* and *is_a* related terms. The IA enhanced client entered terms with its *synonym* and *is_a* terms.

```
<Financial_Investments rdf:ID="Bond">
   <Is_a rdf:datatype="&xsd;string">Instrument</Is_a>
   <Synonym rdf:datatype="&xsd;string">Share</Synonym>
   <Synonym rdf:datatype="&xsd;string">Stock</Synonym>
</Financial_Investments>
<owl:Class rdf:ID="goverment_bond">
   <rdfs:subClassOf rdf:resource="#bond"/> </owl:Class>
```

Figure 14. Created the Financial_Investment Ontology in Protégé 3.1 and IA Used it for Retrieve relations between Terms.

4.2.2 Creating Semantic Web Services on OWL-S Editor in Protégé 3.1

This OWL-S Editor aims to create an easy-to-use editor for creating OWL-S ontology files for web services. The editor is still being developed and the editor is a Tab Widget plug-in for Protégé. Users who are familiar with Protégé, especially using the OWL Plug-in, should have no problem using the OWL-S plug-in. Their tab provides an excellent overview of the main instances of an OWL-S service, providing editing modes for its four main classes; Service, Profile, Process and Grounding.

The OWL-S Editor will be able to load WSDL files [15]. OWL-S services have inputs, outputs, preconditions and results, and these are typically referenced in several places throughout the service description.

An IOPR Manager makes it easy to handle the sometimes complicated relations between these instances, and to create, edit and delete new inputs, outputs, preconditions and results.

The IA project used OWL-S Editor for creating necessary four main classes which are Service, Profile, Process and Grounding. Also, used for create new inputs, outputs, preconditions and results.

The IA used the inputs and outputs of the Profile classes of the SWS and then compare them with client inputs and outputs. For example, below figure is showing that:

```
<profile:Profile rdf:ID="BondsOnlineProfile">
  <profile:hasInput rdf:resource="#Bond"/>
  <service:presentedBy rdf:resource="#BondsOnlineService"/>
  <profile:has_process rdf:resource="#BondsOnlineProcess"/>
  <profile:hasOutput rdf:resource="#Price"/>
</profile:Profile>
```

Figure 15. A Part of the Bonds Online Service in OWL-S Editor and its Profile Information

4.3. StartIA.java Class

There are four ontology files were created and which are located on local host which are called ***Adhesive.owl, Financial_Investments.owl, Vehicle.owl, Software.owl*** (see Appendix K, L, M and N). The StartIA class accesses all above ontology files for creating an empty ontology model and also, checking which ontologies are including entered term by the client. (Please see Figure 12 and Appendix G).

4.4. SetClientKeyWords.java Class

The class tries to send /receive parameters between JSP files and IA classes. The purpose of the class is the required parameters for performing all functions of IA are carrying through the SetClientKeyWords class (Please see Appendix H).

```
 public String getMessage() {
    return message;
}
 public void setMessage(String s){
    temp=s;
    message = s;
 }
```

Figure 16. SetClientKeyWords performs send /receive parameters between JSP files and itself

4.5. FindRelatedTerms.java Class

The class tries to find all “*synonym*” relations and also “*is_a*” relations for the entered client terms. The class tries to read all above ontologies and looking for “*synonym*” or “*is_a*” relations according to client keyword and then IA shows all retrieved related terms to client back.

The below figure is showing the IA reads all the owl files and then check all individuals and after that the IA is try to find all Data Type Properties and checking them *synonym* or *is_a* relations. If, the IA found any *synonym* or *is_a* related term for the client request word than the IA shows all related term as a result (see Appendix I).

```
public List FindIs_ATerms(String SelOntFile,String [] ClientKWL) {
   List is_aList=new LinkedList();
   InputStream in = FileManager.get().open( SelOntFile );
   OntModel model = ModelFactory.createOntologyModel();
   model.read(new InputStreamReader(in), "");
   // model.write(System.out);
   for(int i=0; i<ClientKWL.length; i++){
      Iterator it = model.listIndividuals();
      while (it.hasNext()) {
         Individual individual = (Individual) it.next();
```

```
            System.out.println("individual: " + individual.getLocalName());
            if(individual.getLocalName().equalsIgnoreCase(ClientKWL[i])){
               Iterator propertyIt = model.listDatatypeProperties();
               while (propertyIt.hasNext()) {
                  DatatypeProperty prop = (DatatypeProperty) propertyIt.next();
                  if (prop.getLocalName().equalsIgnoreCase("Is_a")){
                     System.out.println(" datatype property: " + prop.getLocalName());
                     Iterator valueIt = individual.listPropertyValues(prop);
                     while (valueIt.hasNext()) {
                        RDFNode value = (RDFNode) valueIt.next();
                        String str=value.toString();
                        int intPos = str.indexOf("^");
                        String strTemp = str.substring(0, intPos);
                        System.out.println("    value: " + strTemp);
                        is_aList.add(strTemp);
                                          }//3.while
                                    }//secon if
                                 }//2.while
                             }//first if
                        }//1.while
             }//first for
    return is_aList;
     }
```

Figure 17. SetClientKeyWords performs send /receive parameters.

4.6. IA_Match.java Class

The part is main part of matchmaking step of the system. Matchmaking step is performed in this class. There are 23 different functions and they are performed by the class (Please see Appendix J).There are two constructor functions.

- **public IA_Match()**
- **public IA_Match(String url)**

Also, there are 7 different function is responsible for send/receive some necessary parameters between the class and JSP files.

The **public OntModel SelOnt()** function is used for reading owl files and then create and ontology model for the IA project.

The **public List LstAllClasses(OntModel loaded_model)** function is using for listing all classes structure of an owl file.

The **public String getRelation(String ci, String swsi)** function is using for retrieve a relation between client I/O and SWS I/O information. The function returns a string which is including finding relation as a result (Please see Figure 18).

```
 public String getRelation(String ci, String swsi) {
              ci=ci.toLowerCase();
              swsi=swsi.toLowerCase();
              String str="";
              OntModel loaded_model=SelOnt();
              List allClassesList=new LinkedList();
              List alllist= new LinkedList();
              alllist=LstAllClasses(loaded_model);
              for (int f=0; f<alllist.size(); f++){
                 String stra=alllist.get(f).toString();
                 stra=stra.toLowerCase();
                 allClassesList.add(stra); }
              if(ci.equalsIgnoreCase(swsi)) str="EXACT";
              else
if((allClassesList.contains(ci)==true)&&(allClassesList.contains(swsi)==true)) {
                 List getParentList=new LinkedList();
                 List getPrntList=new LinkedList();
getPrntList=getParentClass(loaded_model,ci);//look client  input parent classes
            for (int p=0; p<getPrntList.size(); p++){
                    String strr=getPrntList.get(p).toString();
                     strr=strr.toLowerCase();
                     getParentList.add(strr); }
                 List SubClassesList=new LinkedList();
                 List SubClsList=new LinkedList();
                 SubClsList=getChildClass(loaded_model,ci);
                 for (int c=0; c<SubClsList.size(); c++){
                    String strc=SubClsList.get(c).toString();
                    strc=strc.toLowerCase();
                    SubClassesList.add(strc);}
                 if(getParentList.contains(swsi)==true) str="PLUGIN";
                 else if(SubClassesList.contains(swsi)==true)  str="SUBSUME";
                 else str="FAIL";               } else str="FAIL";
              //   System.out.println(str);
             return str;           }
```

Figure 18. The **getRelation** function performs to find a relation between client I/O and SWS I/O.

The **public int getScore (String rel)** function is used to find a score for a retrieved relation between client I/O and SWS I/O information. The function returns an integer finding score as a result (Please see Figure 19).

```
public  int getScore(String rel){
          int val=0;
          if (rel=="EXACT") val=3;
          else {
```

```
        if (rel=="PLUGIN") val=2;
        else {
           if (rel=="SUBSUME") val=1;
           else {
              if (rel=="FAIL") val=0;
           }
              }
                 }
    return val;
    }
```

Figure 19. The **getScore** function performs to find score for retrieved relation by IA.

The figure 19 shows the score for four different types of relations. As we mention before those related rules for the Matchmaking algorithm in second chapter of the report. Those relational rules are classified as: **EXACT, PLUGIN, SUBSUME, FAIL.** Each relational rule has a score which is shown in Figure 19.

The *Exact* relation has highest score that is “3”. Because client input/output tem is same with the SWS input/output information. The other degree is *Plug-in* and its score is “2”. The plug-in is next best level degree for relation measurement. That mean is the client input/output subsumes the SWS input/output information. Namely, the returned output is most probably can be use instead the requesters expect. The degree of *Subsume* is reverse of the *Plug-in* degree and it is scored as “1”. The last one is *Fail* that means there is no relation between client and SWS input/output terms and it is scored as “0”.

The **public List runMatchForInputs (String [] ClientInp)** function is used to make a list for retrieved scores for inputs information of the client and SWS.

Each client input is compared with all inputs of a returned related SWS OWL-S file by the UDDI and calling some necessary functions to get the scores between inputs. The function returns a list all appointments and finding score as a result between inputs. The output information comparison is performing same as input comparison by IA.

```
public  List runMatchForInputs(String[] ClientInp){
            float scoreinp=0;
            float eachswsinpscr=0;
            int step=1;

              for(int k=0; k<SWSFiles.length; k++) {

                 IA_Match o=new IA_Match(new java.io.File(SWSFiles[k]).toURI().toString());
                 InputList inputList=profile.getInputs();
                 showinp.add(step+".Semantic Web Service");
                 step=step+1;

                 for(int j=0; j<ClientInp.length; j++) {

                    for(int i=0;i<inputList.size();i++) {

                       String relinp=getRelation(ClientInp[j],inputList.inputAt(i).getLocalName());
                       eachswsinpscr=getScore(relinp);
                       scoreinp=scoreinp+eachswsinpscr;
                       eachswsinpscore.add(Float.toString(eachswsinpscr));//each   comparison   result
score
                       eachswsinp.add(inputList.inputAt(i).getLocalName());//each sws input list
                       eachswsclientinp.add(ClientInp[j]);//client inputs list
                       showinp.add(ClientInp[j]+"-->"+inputList.inputAt(i).getLocalName()+"           :
Retrieved Input Relation is= "+relinp+"   Their Score= "+eachswsinpscr+
                                   " And now total score for  "+   SWSFiles[k]+" SWS is : "+scoreinp);
                    }//3.for

                       }//2.for

                 eachswstotinpscore.add(Float.toString(scoreinp/ClientInp.length));
                 showinp.add("INPUTS SCORE WEIGHT: "+scoreinp/ClientInp.length);
                 scoreinp=0;
                                   }//1.for

                 return showinp;
               }
```

Figure 20. The IA is comparing the all client inputs with each SWS inputs.

CHAPTER V

CASE STUDIES OF ISWSF

5.1. The IA is working on Financial Investments Ontology

In this section, the use of user request for documents on “bond” is taken into consideration in the following semantic enhancement and matchmaking steps are carried out in providing the client with proper semantic information by IA.

5.1.1 Semantic Enhancement of Client Input Terms

As the web services search for the terms, results become more and more accurate. As a result the client retrieves all services, related to its request, available in UDDI. UDDI can do only syntactic based search operation so couldn’t find related web services in its registries.

The user enters some term(s), using these term(s) system tries to find ontologies that include these terms. The client should select most relevant ontology to service search operation. In order to help ontology determination, the engine matches the user’s keywords to the ontologies and presents appropriate ontologies. For example “Bond” terms can be found in both financial trading and adhesives ontologies.

At this point, the system uses the client to determine which ontology is most relevant to its required service. Some “*synonym”* keywords and some related keywords such as “*is a*”, are produced according to the terms entered by the user.

The terms entered by the user and their synonym as well as the related terms are passed to UDDI search registries.

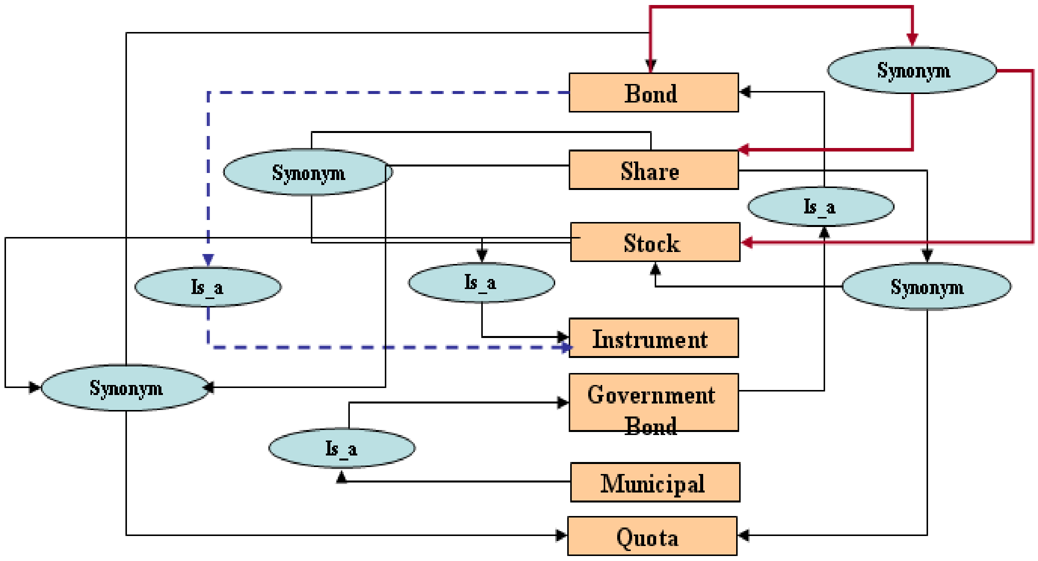

Figure 21. A Part of the Financial Investment Ontology.

Figure 21 illustrates part of financial investment ontology and underlines how relation terms of financial investment ontology are developed.

The system uses this ontology as it makes it more understandable and describes behavior of the system. For example, if the client input is the "bond" and IA wants to check which ontologies contain the term “bond” in them, furthermore the IA wishes to displaying these results to the client for the purpose of reducing the search area which is shown step by step tasks of IA;

Below figure is showing that the main page of the IA. The IA interacts with client and asking the how many keywords will you enter for performing search operation.

Figure 22. Client enter the number of keywords

Then the IA show the keyword enter page to the client for entering his/her keywords from the page, please see Figure 23.

Figure 23. The client enters “Bond” term.

As shown in Figure 24, the term “Bond” can appear in two different ontologies, Financial Investment Ontology and Adhesives Ontology. After that the system will show the names of those ontologies and the system will ask to client which ontology is relevant with your search. Assume that the client choosed the financial investment ontology. Anymore the system will operate on only that ontology which is shown in next figure.

Figure 24. The IA found the relevant ontologies of the term “Bond”.

After all, the agent is checking the "Bond" term in Financial_Investments ontology for finding its *synonym* terms or *is_a* related terms (See Figure 25).

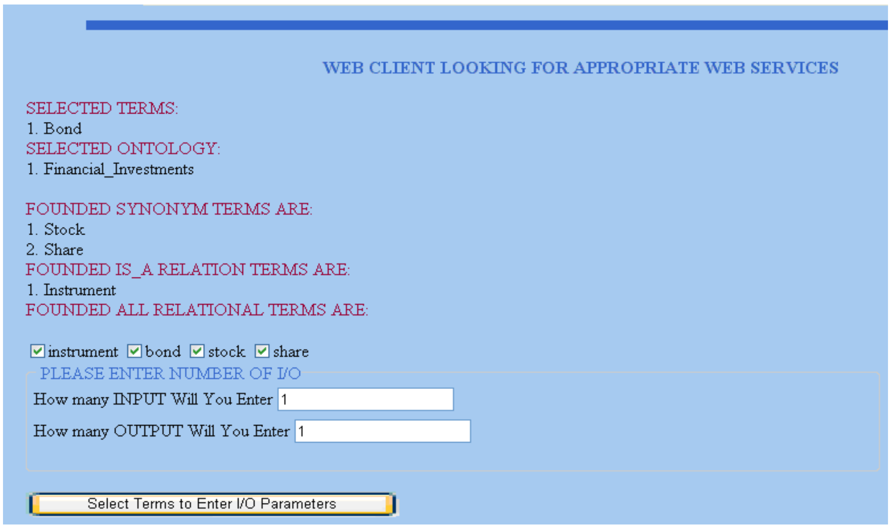

Figure 25. The IA is showing the results of the "Bond" term’s *synonym* and *is_a* related terms.

As shown in Figure 25, the system specifies that the term "Bond" has *synonym* "Stock" and “Share", after finding synonym term(s) of user key term. Also the system found that the term “Bond” has an *Is_a* relation that is “Instrument”. The means is “bond is an instrument”. After specifying the synonyms and Is_a relations, the agent prepares to enhance search criteria and begins searching and sending those related terms to UDDI for retrieve related web services.

The system applies matchmaking step after searching UDDI. This matchmaking is done for all returned web services related to the client’s criteria. The next section describes how the system applies matchmaking algorithm.

5.1.2 Matchmaking Step applied on Financial Investments Ontology

Matchmaking algorithm was developed to show how and why matchmaking is applied between enhanced client terms with SWS terms. UDDI return SWS addresses and IA connect those related SWS files and IA try to extract profile data of each SWS and the client I/O request then matchmaking algorithm is applying to define relation between them. Each SWS service profile(s) stores IOPE (Input, Output, Precondition and Effect) information of the web service.

The Input of the client information and that of the UDDI provided SWS profile(s) as well as the output of the client information and the output of the UDDI provided SWS profile(s) are matched to each other, making the task of matchmaking logical and easy to implement.

The system extracts the respective IOPE and attempts to apply matchmaking algorithm in order to find the degree of a match between client request and UDDI provided SWS profile(s).

Figure 26 shows a simple example illustrating how a client request for a service is matched with a Bonds Online Service of SWS. In this figure the service gives price information of Bonds and the input of the service is restricted by instances of the concept Bond as defined in Bonds Online Service ontology and the service generates outputs as Price in Bonds Online Service ontology.

```
<profile:Profile rdf:ID="BondsOnlineService">
    <profile:serviceName>BondsOnlineProfile</profile:serviceName>
        <input>
<profile:ParameterDescription rdf:ID="Bond">
<profile:ParameterName>Bond</profile:ParameterName>
<profile:restrictedTo rdf:resource="Fin_Inv.owl#Bond"/>
</profile:ParameterDescription>
        </input>
<output>
<profile:ParameterDescription rdf:ID="Price">
<profile:ParameterName>Price </profile:ParameterName>
<profile:restrictedTo rdf:resource="Fin_Inv.owl#Price"/>
</profile:ParameterDescription>
        </output>
</profile:Profile>
```

Figure 26. I/O information of Bonds Online Service

Figure 27 shows a possible client request for a service, and is expressed in the same format of the Bonds Online Service. The client input is instances of Bond and its output is an instance of the term “Price”.

```
<profile:Profile rdf:ID="RequestFor BondsOnlineService ">
    <profile:serviceName> Bonds Online Service </profile:serviceName>
        <input>
<profile:ParameterDescription rdf:ID="Bond">
<profile:ParameterName>Bond</profile:ParameterName>
<profile:restrictedTo rdf:resource="#Bond"/>
</profile:ParameterDescription>
        </input>
<output>
<profile:ParameterDescription rdf:ID="Price">
<profile:ParameterName>Price</profile:ParameterName>
<profile:restrictedTo rdf:resource="#Price"/>
</profile:ParameterDescription>
        </output>
</profile:Profile>
```

Figure 27. I/O information of client request for Bonds Online Service.

Below figure is how the client enters his/her input and output terms for beginning of matchmaking algorithm. The IA will enter all returned SWS files from UDDI and then compare the term “Bond” is exist in SWS file as input. If exists the

IA will specify the relation between inputs are EXACT, otherwise the IA will check financial investment ontology for describing the relation between inputs. There are three possibility as we mention before; PLUG-IN, SUBSUME or FAIL. But in this example, the relation will be EXACT since SWS and Client inputs are same. This process is applying as well as for describing outputs relations between Client outputs and SWS outputs.

Figure 28. Client enters I/O information.

The matching operation needs a match between the inputs and outputs information of client and the SWS respectively. In this scenario the inputs of the client and the SWS are same so their match is termed as exact match according to matchmaking algorithm.

Furthermore, the client's output is the term "Price", whereas the SWS output is "Price". So the relation will define as exact match by the IA as well. Finally, the IA recognizes that the client request and SWS are related to each other because exact match was found for both their inputs and outputs information.

5.1.3 Result of Case Study for Financial Investments Ontology

The case study was performed by system's processing a client request and providing the client with SWS that can service that particular request. This case study the client request contained the term "Bond". The system first attempted to find ontologies that contain, or are related to, the term "bond". In this attempt the system found two ontologies, "Financial Investment" ontology and "Adhesives" ontology, containing/related to the requested term (bond). After discovering that two above ontologies for the specific request the IA query's the client what ontology interests it. The possible answer "Financial Investment" is chosen by the client and this causes the IA to investigate all synonyms of the requested term "bond" in that ontology. Finally, having found the related terms and synonyms of the requested term "bond" the IA looks up the UDDI to locate all/any web services that exist in its registry. This is done to increase the probability of finding a suitable service in the UDDI registry for the client's request. For instance, one suitable web service found for "bond" term was "Bonds Online Service". IA now matches the input and output parameters of the client to that of the web services. In this case study the degree of match, for the client requested term "Bond" with the IA discovered service "Bonds Online Service", was found to be "Exact". Finally, the system understands that this service is suitable for the client request.

Below figure is showing the returned match results of after applying matchmaking step by IA for the client request. For example, client term "bond" and SWS input term "Bond" and their relation is matched as EXACT in third Semantic Web Service of the below figure. Then IA specifies their score as highest score "3.0". The outputs are the term "Price" for both and same match is found as "exact" as well. So the IA decides the service is BEST according to client request. This service name is Bonds Online Quotes Service which is shown as third SWS in figure 30.

```
RESULTS
IA RETURNED MATCH SCORES FOR INPUTS:
1.Semantic Web Service
bond-->Bond : Founded Input Relation is= EXACT Their Score= 3.0 SWS TotalScore is : 3.0
bond-->Date : Founded Input Relation is= FAIL Their Score= 0.0 SWS TotalScore is : 3.0
INPUTS SCORE WEIGHT: 1.5
2.Semantic Web Service
bond-->BookInformation : Founded Input Relation is= FAIL Their Score= 0.0 SWS TotalScore is : 0.0
INPUTS SCORE WEIGHT: 0.0
3.Semantic Web Service
bond-->Bond : Founded Input Relation is= EXACT Their Score= 3.0 SWS TotalScore is : 3.0
INPUTS SCORE WEIGHT: 3.0
4.Semantic Web Service
bond-->Name : Founded Input Relation is= FAIL Their Score= 0.0 SWS TotalScore is : 0.0
bond-->Job : Founded Input Relation is= FAIL Their Score= 0.0 SWS TotalScore is : 0.0
INPUTS SCORE WEIGHT: 0.0
5.Semantic Web Service
bond-->Symbol : Founded Input Relation is= FAIL Their Score= 0.0 SWS TotalScore is : 0.0
bond-->Stock : Founded Input Relation is= FAIL Their Score= 0.0 SWS TotalScore is : 0.0
INPUTS SCORE WEIGHT: 0.0
IA RETURNED MATCH SCORES FOR OUTPUTS:
1.Semantic Web Service
price-->Price : Founded Output Relation is= EXACT Their Score= 3.0 SWS TotalScore is : 3.0
OUTPUTS SCORE WEIGHT: 3.0
2.Semantic Web Service
price-->Price : Founded Output Relation is= EXACT Their Score= 3.0 SWS TotalScore is : 3.0
OUTPUTS SCORE WEIGHT: 3.0
3.Semantic Web Service
```

Figure 29. IA returned results of matching process between client I/O and SWS I/O.

The IA performs many functions for finding closest web service for the client. For this purpose the IA interact many times with client and SWSs. Also IA is giving some comments to the client as BEST, WORST or Nearly BEST etc... Please see Figure 31. The third service retrieved the BEST service according to client request and the IA is giving the access point as web address. If client found the service just click on the address and may gets required service.

1. SWS is:
Inputs Similarity: 50.0
Outputs Similarity: 100.0
Average Similarity: 75.0
THIS SWS ACCESS POINT IS : http://www.mkts.com/SPSIndex.htm
COMMENTS FOR SERVICE: Bond Price Infomation Service
Your Input/Output(s) are exist but there is extra I/O information:FINE Enough!

2. SWS is:
Inputs Similarity: 0.0
Outputs Similarity: 100.0
Average Similarity: 50.0
THIS SWS ACCESS POINT IS : http://www.threepines.net/buybook1/buybook1.asmx
COMMENTS FOR SERVICE: This service is selling book in stock.
Not GOOD!

3. SWS is:
Inputs Similarity: 100.0
Outputs Similarity: 100.0
Average Similarity: 100.0
THIS SWS ACCESS POINT IS : http://www.bondsonlinequotes.com/
COMMENTS FOR SERVICE: Stock and bond prices, bond yields, historical stock market quotes, stock and bond indexes from around the world are at your fingertips: data and descriptive information are continually updated for each security, with up to 30 years history for many securities. BEST!

4. SWS is:
Inputs Similarity: 0.0
Outputs Similarity: 0.0
Average Similarity: 0.0
THIS SWS ACCESS POINT IS : http://www.ultrashare.com
COMMENTS FOR SERVICE: UltraShare is a collaborative network of over 300,000 consultants and 5,000 business partners set to fill contract I.T. job openings within 48 hours in all IT skillsets.
WORST!

Figure 30. IA returned Best SWS for client after matching process.

5.2. The IA is working on Software Ontology

The second part of the case study section, the use of user request for documents on the term “java” and “questionnaire” is taken into consideration in the following semantic enhancement and matchmaking steps are carried out in providing the client with proper semantic information by IA. The client is looking for a service that is related with java questionnaire service for measuring of java knowledge of the web service users.

5.2.1 Semantic Enhancement of Client Input Terms

In this example the client is entering the terms “java” and “questionnaire” as a search keywords. The IA starts to find which ontologies are including the terms “java” and “questionnaire”. The IA found the terms “java” and “questionnaire” in software ontology. Some *synonym* keywords and some related keywords such as “is a”, are produced according to the terms “java” and “questionnaire”.

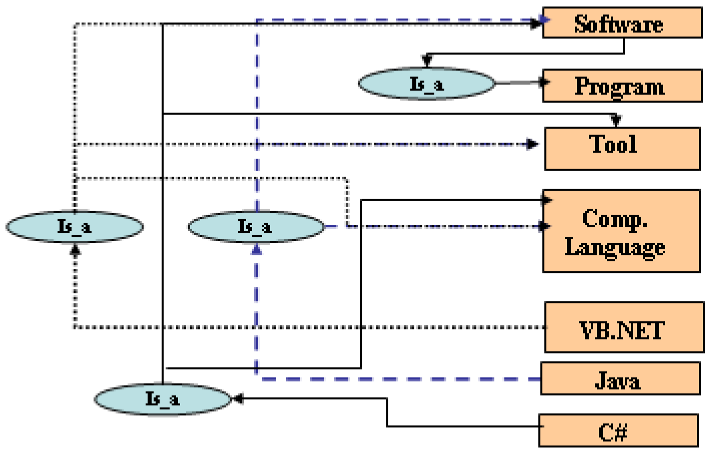

Figure 31. A Part of the Software Ontology.

Figure 31 illustrates part of software ontology and underlines how relation terms of software ontology are developed. The IA interacts with client and asking the how many keywords will you enter for performing search operation (see Figure 32).

The system starts to find a required java test or questionnaire web service by the client. The client will enter some keyword(s) for retrieve related ontology and then the IA will continue to operate semantic enhancement of client input terms and matchmaking step tasks on that ontology.

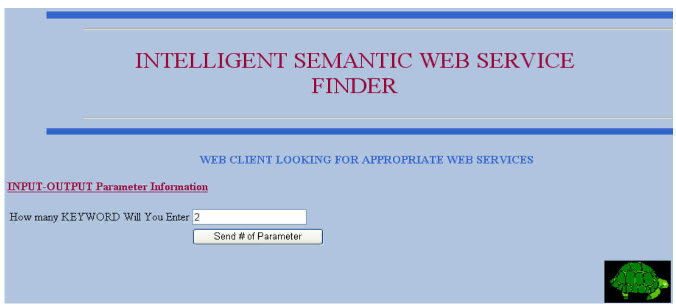

Figure 32. The IA asked to client how many keywords will enter by the client.

Then the IA show the keyword enter page to the client for entering his/her keywords from the page (see Figure 33). Assume that the "java" and "questionnaire" terms are entered by the user. Now, the IA will check which ontologies contain the terms "java" and "questionnaire" in them, furthermore the IA wishes to display these results to the client for the purpose of reducing the search area.

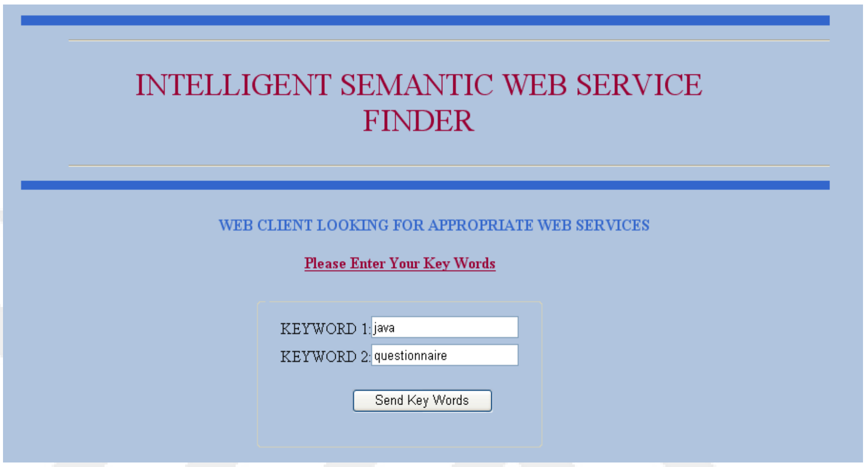

Figure 33. The client enters "Java" and "questionnaire" terms.

The system found the terms "Java" and "questionnaire" can appear in software ontology (shown in Figure 34). Assume that the client selected the Software ontology. Anymore the system will operate on only that ontology.

Figure 34. The IA found the relevant ontology of the user terms.

After all, the agent is checking the “Java” and “questionnaire” terms in Software ontology for finding its *synonym* terms or *is_a* related terms which is shown as a result in Figure 35.

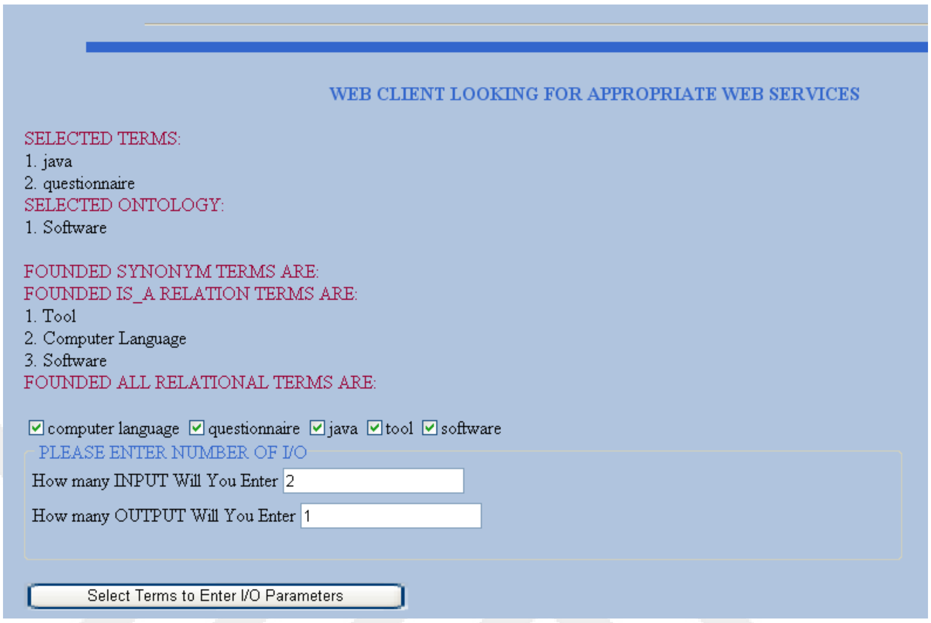

Figure 35. The IA is showing the results of retrieved related terms.

As shown in Figure 35, the system specifies that the terms "java" and “questionnaire” has no synonym terms however, the system found that the terms "java" and “questionnaire” have some Is_a relation that are “tool”,”computer language” and “software”. After specifying all synonyms and Is_a relations, the agent prepares to enhance search criteria and begins searching and sending those related terms to UDDI for retrieve related web services. So, the retrieved *synonym* or *is_a* terms are passed to UDDI search registries. The system uses the Software ontology as it makes it more understandable and describes behavior of the system.

The system applies matchmaking step after searching UDDI. This matchmaking is done for all returned web services related to the client’s criteria. The next section describes how the system applies matchmaking step on the Software ontology.

5.2.2 Matchmaking Step applied on Software Ontology

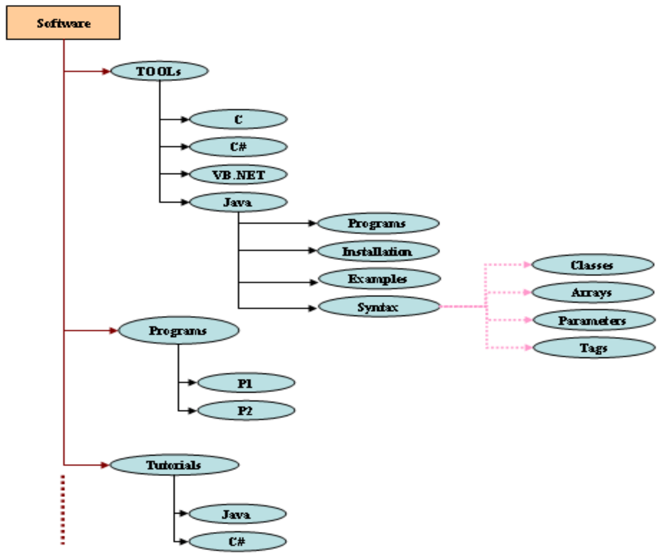

Figure 36. A part of Software Ontology.

The Software ontology is developed in protégé 3.1 and the ontology is including classes, subclasses, individuals and data type properties about computer software. Above figure is showing a part of Software ontology (see Figure 36).

Figure 37 shows a simple example illustrating how a client request for a service is matched with a Java Questionnaire Service of SWS. In this figure the service gives a tutorial about a java questionnaire.

```
<service:Service rdf:ID="JavaQuestionaireService">
      <service:presents>
        <profile:Profile rdf:ID="JavaQuestionaireProfile">
          <profile:hasInput>
            <process:Input rdf:ID="questionnaire"/>
          </profile:hasInput>
          <service:presentedBy rdf:resource="#JavaQuestionaireService"/>
          <profile:hasInput>
            <process:Input rdf:ID="java"/>
          </profile:hasInput>
          <profile:hasOutput>
            <process:Output rdf:ID="result"/>
          </profile:hasOutput>
        </profile:Profile>       </service:presents>
```

Figure 37. I/O information of Java Questionnaire Service.

Assume that the client inputs are “java” and “questionnaire” and his/her output is “result”. Below figure is how the client enters his/her input and output terms for beginning of matchmaking step. The IA will enter all returned SWS files from UDDI and then compare the terms “java” and “questionnaire” are exist in SWS file as input. If exists the IA will specify the relation between inputs are EXACT, otherwise the IA will check Software ontology for describing the relation between inputs. Possible relation rules are PLUG-IN, SUBSUME or FAIL. But in this example, the relation will be two EXACT for inputs. Because “java” and “questionnaire” terms are same in SWS and Client inputs. So each exact relation degree is 3 and we have two exact relations so the score will be 6. The IA will calculate the similarity percentage for input terms and output terms. So, the IA will calculate the similarity percentage as %100. Another scenario is if the client looking for more specific Java. NET or Net beans 4.1 questionnaire service. So, the IA will check Java. NET or Net beans 4.1 is subsumed by java class in the software ontology. If exists then the IA will understand the relation is Plug-in and it is scored as 2. This process is applying as well as for describing outputs relations between Client outputs and SWS outputs.

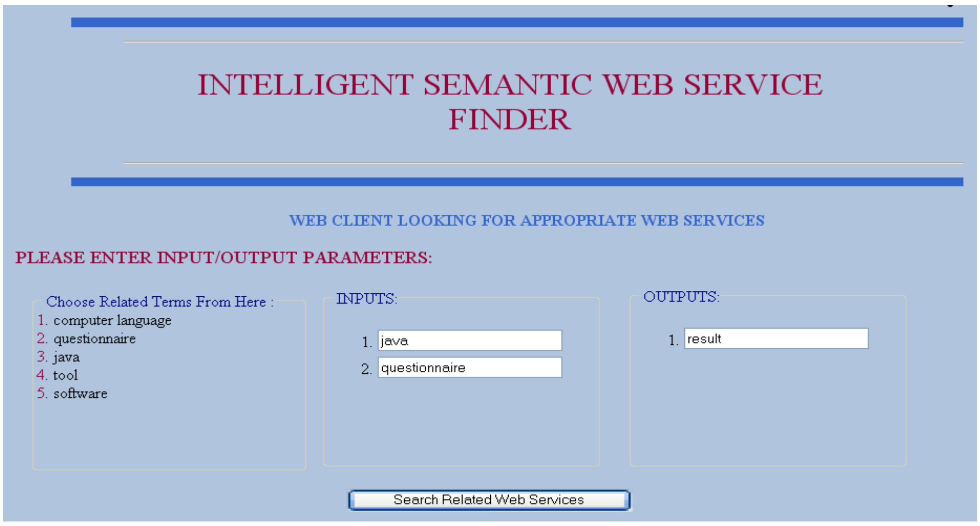

Figure 38. Client enters I/O information.

5.2.3 Result of Case Study for Software Ontology

In this case study the client request included the terms “java” and “questionnaire”. They system first tried to find ontologies that contain, or are related to, the terms “java” and “questionnaire”. In this effort the system found just one ontology that is “Software” ontology, containing/related to the requested term (java and questionnaire). After discovering that ontology for the specific request, assume that the client is selected the “Software Ontology” and this grounds the IA to examines all synonyms of the requested terms “java” and “questionnaire” in that ontology.

Finally, having found the related terms and synonyms of the requested terms “java” and “questionnaire” the IA looks up the UDDI to locate all/any web services that exist in its registry.

Below figure is showing the returned match results of after applying matchmaking step by IA for the client request. For example, client terms “java” and “questionnaire” and SWS input terms “java” and “questionnaire” and their relation is matched as EXACT in second Semantic Web Service of the below figure. Then IA specifies their score as highest score “3.0”.

The outputs are the term “Result” for both and same match is found as “Exact” as well. So the IA decides the service is BEST according to client request. This service name is Java Questionnaire Service which is shown as second SWS in figure 41.

Figure 39. IA returned results of matching process between client I/O and SWS I/O.

The IA is performed matchmaking step for input terms of client and SWS. Below figure is showing the input score that is 3.0.

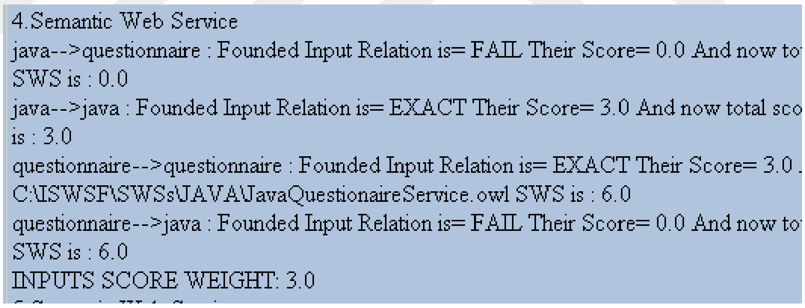

Figure 40. IA found fourth SWS is most similar client I/O.

The IA performs many functions for finding closest web service for the client. For this purpose the IA interact many times with client and SWSs. Also IA is giving some comments to the client as BEST, WORST or Nearly BEST etc... (Please see Figure 41).

The fourth service retrieved the BEST service according to client request and the IA is giving the access point as web address. If client found the service just click on the address and may gets required service.

4. SWS is:
Inputs Similarity: 100.0
Outputs Similarity: 100.0
Average Similarity: 100.0
THIS SWS ACCESS POINT IS : http://www.developer.com/java/ent/article.php/630361 BEST!

5. SWS is:
Inputs Similarity: 66.666664
Outputs Similarity: 0.0
Average Similarity: 33.333332
THIS SWS ACCESS POINT IS : http://xml.apache.org/security/Java/installation.html

Figure 41. IA returned Best SWS for client after matching process.

CHAPTER VI

CONCLUSIONS

The main problems of current web service technologies are the shortage of semantic parts, increasing number of web services in the web, and syntactic-based search operation for web services. These problems make discovering of appropriate web services challenging.

This thesis provides an implementation of an intelligent agent that can be used to find related Semantic Web Services based on a client's request. This IA performs two main functions, namely, (i) enhancing client request term with synonym or related terms and (ii) matching the SWS I/O and client I/O. The agent specifies and shows the results of its findings as a page, including results of related web services, showing scores of the matchmaking step.

The purpose of the matchmaker algorithm was [6, 7, 8 and 9] to be implemented in UDDI. However, UDDI being a well-established simple standard, it should not be subjected to change. Hence, we propose that the IA should be used for this purpose. The agent can determine the services appropriate for the client and suggests the determined services to the client. The agent also considers the functionalities of web services and communicates these functionalities/operations to the client. This way the agent performs useful decision making process for the client.

The proposed system is implemented in the JAVA platform by using NETBEANS 4.0. Also, the Financial Investments, Vehicle, Software and Adhesive ontologies are created in Protégé 3.1. Some semantic web services are created for our test applications in Protégé 3.1 as well. The client sends some request parameters to the system such as the term "Bond" and the system takes the term. The system checks the term “Bond” in our predefined ontologies. Then the system sends to the client all retrieved ontology names that include the term “Bond”. The client selects the most relevant ontology name according to his/her search criteria. The system works only on selected ontology from then on. The system accesses the selected ontology and extracts all related terms such as *synonym* terms and *is_a* related terms according to the term(s) entered by the client. Then all terms are sent to the UDDI and the system receives all semantic web services information and their addresses. The system asks for input/output parameters from client after that connects to all those returned web services and extracts profile information by accessing their OWL-S file(s) or their semantic descriptions. Finally, the system starts to perform the matchmaking algorithm between all input/output parameters of client and input/output parameters of returned web services. The system is able to specify some relationships between those input/output parameters such as EXACT, PLUGIN, SUBSUME and FAIL. According to those relationships the system serves appropriate web services to the client based on client request.

This intelligent system may as well be applied on e-learning. For example, the system can be used for searching, selecting and registering courses/modules. By finding, selecting and combining these services, the system can decrease the user interaction in these services for e-learning management. We just used two properties for semantic enhancement of client input terms. As was mentioned earlier these properties are *synonym* and *is_a*. These properties might change depending on different cases. If we increase the number of properties the performance of the system will improve. Thus, the IA will be able to get better results and elect better web services.

REFERENCES

[1] http://www.w3.org/TR/soap/ , Last Visited Page: 18/07/2005.

[2] http://www.w3.org/TR/wsdl.html , Last Visited Page: 18/07/2005.

[3] http://www.uddi.org/, Last Visited Page: 18/07/2005.

[4] http://owlseditor.semwebcentral.org/, Last Visited Page: 18/07/2005.

[5] Roger H.L. Chiang, Cecil Eng Huang Chua and Veda C. Storey, A smart web query method for semantic retrieval of web data, Data & Knowledge Engineering, Volume 38, Issue 1, July 2001,Pages63-84, Last Visited Page:20/07/2005, *http://www.sciencedirect.com/science*http://www.kns.dlr.de/Projects/spreading/spreading190198.html

[6] K. Sycara, M. Paolucci, A. Ankolekar and N. Srinivasan, "Automated Discovery, Interaction and Composition of Semantic Web services," in *Journal of Web Semantics*, Volume 1, Issue 1, September 2003, pp. 27-46.

[7] N. Srinivasan, M. Paolucci, and K. Sycara, "Adding OWL-S to UDDI, implementation and throughput," in *First International Workshop on Semantic Web Services and Web Process Composition **(SWSWPC 2004*) 6-9, 2004, San Diego, California, USA.

[8] M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara, "Importing the Semantic Web in UDDI," in *Proceedings of Web Services, E-business and Semantic Web Workshop*, 2002.

[9] T. Kawamura, J. A. De Blasio, T. Hasegawa, M. Paolucci, and K. Sycara, "A Preliminary Report of a Public Experiment of a Semantic Service Matchmaker combined with a UDDI Business Registry," in *1st International Conference on Service Oriented Computing (ICSOC 2003)*, Trento, Italy, December 2003.

[10] http://protege.stanford.edu/download/registered.html, Protégé 3.0 of Stanford University, Last Visited Page: 15/07/2005.

[11] http://www.hpl.hp.com/semweb/jena.htm , Last Visited Page: 18/07/2005.

[12] http://www.mindswap.org/2004/owl-s/api/index.shtml, Last Visited Page: 18/07/2005.

[13] http://java.sun.com/products/jsp/overview.html, Last Visited Page: 18/07/2005.

[14] http://protege.stanford.edu/, Last Visited Page: 18/07/2005.

[15] http://owlseditor.semwebcentral.org/, Last Visited Page: 18/07/2005.

[16] Stefan Tang, a MS Thesis for Matching of Web Service Specifications Using DAML-S Descriptions, March 18th, 2004, Last Visited Page: 18/07/2005, http://ivs.tu-berlin.de/Projekte/owlsmatcher/resources/thesis_steftang.pdf ,

[17] N. Srinivasan, M. Paolucci, and K. Sycara, " CODE: A Development Environment for OWL-S Web services," http://iswc2004.semanticweb.org/demos/10/paper.pdf , Last Visited Page: 18/07/2005.

[18] Anna V.Zhdanova, Jena 2.0 tutorial, Last Visited Page: 18/07/2005, http://www.deri.at/teaching/seminars/internal/slides/Jena-2.pdf .

[19] C.K. Olesen, T.Hqjgaard", MS thesis is for a Semantic UDDI Repository Combining OWL-S and WSDL-S, Last Visited Page: 15/07/2005, http://groups.yahoo.com/group/semanticweb/files/**Sem-UDDI**.pdf .

[20] G. Denker and D. Elenius and D. Martin, OWL-S Editor for Creating Semantic Web Services, www.iswc2004.semanticweb.org/demos/02/paper.pdf, Last Visited Page: 15/07/2005.

APPENDICES

APPENDIX A: index.jsp

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<html>
   <body bgcolor="#b0c4de">
      <blockquote>
         <ul>
            <li>
            <form id="form1" runat="server">
                     <hr style="width: 850px; color: #3366cc; font-style: normal; font-family:
Verdana;
            height: 10px; background-color: lightsteelblue; font-variant: small-caps"
align="right" />
               <hr style="width: 800px; height: 2px" align="right" />
              </p>
               <blockquote>
                  <blockquote>
                     <p align="center"><font size="6" color="#990033">
                     INTELLIGENT SEMANTIC WEB SERVICE FINDER</font></p>
                  </blockquote>
               </blockquote>
               <hr style="width: 800px; height: 2px" align="right" />
               <hr style="width: 850px; color: #3366cc; font-style: normal; font-family: Verdana;
            height: 9px; background-color: lightsteelblue; font-variant: small-caps"
align="right" />
               <p align="center">        <b>
               <font color="#3366CC">WEB CLIENT LOOKING FOR APPROPRIATE WEB
               SERVICES </font></b></p>
            </form></li>
         </ul>
      <p><font color="#990033"><u><b>INPUT-OUTPUT Parameter Information</b></u>
      </font>
       </p> <p>
      <form method=GET action="EnterUserKeyWords.jsp">
      <p>       <table>
         <tr><td>How many KEYWORD Will You Enter<td> <input type=text name="num">
         <tr><td> <td><input type=submit value="Send # of Parameter">
      </table>
       
       <img border="0" src="file:///C:/ISWSF/IA.bmp" width="90" height="60"
align="right">
            </blockquote>
   </body>
</html>
```

APPENDIX B: EnterUserKeyWords.jsp

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<html>
   <head>
      <title>
         ISWSF
      </title>
   </head>
   <body bgcolor="#b0c4de">
      <blockquote>
         <ul>
            <li>
            <form id="form2" runat="server">
               <hr style="width: 850px; color: #3366cc; font-style: normal; font-family: Verdana;
        height: 10px; background-color: lightsteelblue; font-variant: small-caps" align="right"
/>
               <hr style="width: 800px; height: 2px" align="right" />
                   <p align="center"><font size="6" color="#990033">
                   INTELLIGENT SEMANTIC WEB SERVICE FINDER</font></p>
               <hr style="width: 800px; height: 2px" align="right" />
               <hr style="width: 850px; color: #3366cc; font-style: normal; font-family: Verdana;
              height: 9px; background-color: lightsteelblue; font-variant: small-caps"
align="right" />
               <p align="center">        <b>
               <font color="#3366CC">WEB CLIENT LOOKING FOR APPROPRIATE WEB
               SERVICES </font></b></p>
            </form>
         </li>
     </ul>
<%      String str = request.getParameter("num");
            int temp = Integer.parseInt(str);  %>       <center>
  <p><font color="#990033"><u><b>Please Enter Your Key Words</b></u>
   </font></p>
<fieldset style="width: 300px; height: 157px; padding: 2">
      <legend></legend>
   <TABLE>
      <form method=GET action="ShowUserKeyWords.jsp">
<%     for ( int i = 0; i < temp; i++ ) {    %>
      <TR>
  <TD>
KEYWORD <%= i+1 %>:<input type=text name="<%=("k"+Integer.toString(i+1))%>">
</TD>
     </TR>        <%    } %>
   </TABLE>
   <tr>
<td>
<br>
      <td><input type=submit value="Send Key Words"> </legend>
</center>
     <jsp:useBean id="SetClientKeyWords" class="IA.SetClientKeyWords" />
      <jsp:setProperty name="SetClientKeyWords" property="message"
value="<%=Integer.toString(temp)%>" />
   <%--    <jsp:getProperty name="SetClientKeyWords" property="message" />--%>
      </blockquote>
   </body>
 </html>
```

APPENDIX C: ShowUserKeyWords.jsp

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<html>
   <head>
      <title>
         ISWSF
      </title>
   </head>
   <body bgcolor="#b0c4de">
      <blockquote>
         <ul>
            <li>
            <form id="form3" runat="server">
               <hr style="width: 850px; color: #3366cc; font-style: normal; font-family: Verdana;
      height: 10px; background-color: lightsteelblue; font-variant: small-caps" align="right" />
               <hr style="width: 800px; height: 2px" align="right" />
               <blockquote>
                  <blockquote>
                     <p align="center"><font size="6" color="#990033">
                     INTELLIGENT SEMANTIC WEB SERVICE FINDER</font></p>
                  </blockquote>
               </blockquote>
               <hr style="width: 800px; height: 2px" align="right" />
               <hr style="width: 850px; color: #3366cc; font-style: normal; font-family: Verdana;
               height: 9px; background-color: lightsteelblue; font-variant: small-caps"
align="right" />
               <p align="center">        <b>
               <font color="#3366CC">WEB CLIENT LOOKING FOR APPROPRIATE WEB
               SERVICES </font></b></p>
            </form>

         </li>
        </ul>
          <%@ page import='com.hp.hpl.jena.rdf.model.*'%>
      <%@ page import= 'com.hp.hpl.jena.ontology.OntModel'%>
      <%@ page import= 'com.hp.hpl.jena.ontology.*'%>
      <%@ page import= 'com.hp.hpl.jena.util.FileManager'%>
      <%@ page import= 'com.hp.hpl.jena.util.iterator.ExtendedIterator.*'%>
      <%@ page import ='java.io.*'%>
      <%@ page import ='java.util.List'%>
      <%@ page import ='java.util.Iterator'%>
      <%@ page import ='java.util.LinkedList'%>
      <%@ page import ='java.util.*'%>
<% IA.SetClientKeyWords NofKW=new IA.SetClientKeyWords();%>
<% IA.StartIA start=new IA.StartIA();%>
<% String tmp=NofKW.getTemp();
int total=Integer.parseInt(tmp);%>
<% start.setTemp(total); %>
<% for (int i = 0; i < total; i++ ) {%>
<% String prm=request.getParameter("k"+Integer.toString(i+1));%>
<% start.setMessage(prm);
%>

<%}%>
     </br>
```

```
<% String[] ClientKWL = new String[total];
ClientKWL =start.PrintClientKWL();%>
<fieldset style="width: 272px; height: 50px; padding: 2">
      <legend><b><font color="#3366CC">Your Term/Terms Are:</font></b></legend>
      <form method=GET action="ShowSelOnt.jsp">
    <%for(int j = 0; j < total; j++ ) {%>
      <%--   <font color="#990033"> <br><%=ClientKWL[j]%></font>--%>
    <TR><TD><input type=checkbox  name="clientTerms" value="<%=ClientKWL[j]%>"
checked><%=ClientKWL[j]%></TD></TR>   <br>
<%}%>
 
</fieldset>
<% List RelatedOntologyList= new LinkedList();
RelatedOntologyList=start.FindRelatedOntologyNames();
if (RelatedOntologyList.isEmpty()){
String Msg="THERE IS NO SUCH KEY TERM/TERMS,Please Choose Another and Try
Again Pls!";%>
 <br><br> <br>
 <font color="#990033"><%=Msg%></font><br> <br> <br> <br>
      <center><A HREF="index.jsp">Back Main Page</A></center>
<%} else {
Set set= new HashSet(RelatedOntologyList);
Iterator itt = set.iterator();
%>
<br><br><b>
 <font color="#990033">Your Term(s) are Encountered Below Ontologies: Which Ontoogy is
Related with your Search?
 </font>
</b>
      <form method=GET action="ShowSelOnt.jsp">
<% while (itt.hasNext()) {
String st =(String)itt.next();%>
<br>
      <TR>
<TD>
<input type=checkbox  name="mycheckbox" value="<%=st%>"><%=st%>
</TD>
</TR>
<%
}//while%>
      <br><br>
<TR>
<TD>
<input type=submit value="Select Ontology Name">
</TD>
</TR>
<%}//else %>
       </blockquote>
   </body>
</html>
```

APPENDIX D: ShowSelOnt.jsp

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<html>
   <head>
      <title>
         ISWSF
      </title>
   </head>
   <body bgcolor="#b0c4de">
      <blockquote>
         <ul>
            <li>
            <form id="form2" runat="server">
               <hr style="width: 850px; color: #3366cc; font-style: normal; font-family: Verdana;
      height: 10px; background-color: lightsteelblue; font-variant: small-caps" align="right" />
               <hr style="width: 800px; height: 2px" align="right" />
               <blockquote>
                  <blockquote>
                     <p align="center"><font size="6" color="#990033">
                     INTELLIGENT SEMANTIC WEB SERVICE FINDER</font></p>
                  </blockquote>
               <hr style="width: 800px; height: 2px" align="right" />
               <hr style="width: 850px; color: #3366cc; font-style: normal; font-family: Verdana;
               height: 9px; background-color: lightsteelblue; font-variant: small-caps"
align="right" />
               <p align="center">        <b>
               <font color="#3366CC">WEB CLIENT LOOKING FOR APPROPRIATE WEB
               SERVICES </font></b></p>
            </form></li>
         </ul>
      <%@ page import='com.hp.hpl.jena.rdf.model.*'%>
      <%@ page import= 'com.hp.hpl.jena.ontology.OntModel'%>
      <%@ page import= 'com.hp.hpl.jena.ontology.*'%>
      <%@ page import= 'com.hp.hpl.jena.util.FileManager'%>
      <%@ page import= 'com.hp.hpl.jena.util.iterator.ExtendedIterator.*'%>
      <%@ page import ='java.io.*'%>
      <%@ page import ='java.util.List'%>
      <%@ page import ='java.util.Iterator'%>
      <%@ page import ='java.util.LinkedList'%>
      <%@ page import ='java.util.*'%>
      <%@ page import ='java.awt.Container.*'%>
      <%@ page import ='javax.swing.*'%>
      <%@ page import ='com.hp.hpl.jena.util.iterator.IteratorIterator.*'%>
      <%@ page import ='com.hp.hpl.jena.ontology.Individual'%>
      <%--CLIENT SELECT RELATED TERMS WITH HIS/HER SEARCH CRITERIA --%>
<% String [] ClientKWL = request.getParameterValues("clientTerms");%>
<FONT size="3" color="#990033">SELECTED TERMS:<BR></FONT>
<%for(int j=0; j<ClientKWL.length; j++){%>
<%=(j+1)%>. <%=ClientKWL[j]%><BR>
<%}%>
      <%--CLIENT SELECT RELATED ONTOLOGY NAME WITH HIS/HER SEARCH
CRITERIA --%>
<% String [] checkboxValue = request.getParameterValues("mycheckbox");%>
      <FONT size="3" color="#990033">SELECTED ONTOLOGY:<BR></FONT>
<%for(int i=0; i<checkboxValue.length; i++){%>
```

```
<%=(i+1)%>. <%=checkboxValue[i]%><BR>
<%}%>

<% String SelOntFile="C:/Documents and
Settings/duygu/Desktop/"+checkboxValue[0]+".owl";%>
       <%--<%=SelOntFile%>--%>
     <% IA.IA_Match  mtch=new IA.IA_Match();
       mtch.setSelOnt(SelOntFile);%>
       <%--CALL A FUNCTION WHICH WILL FIND SYNONYM TERMS --%><%
IA.FindRelatedTerms frt = new IA.FindRelatedTerms();%>

<% List SynonymList= new LinkedList();
SynonymList=frt.FindSynonymTerms(SelOntFile,ClientKWL);
       Set set= new HashSet(SynonymList);
       Iterator synLst = set.iterator();%><br>
       <FONT size="3" color="#990033">RETRIEVED SYNONYM TERMS ARE:</FONT>
<% int i=0;
      while (synLst.hasNext()) {
          String st =(String)synLst.next();%>
       <br>  <%=(i+1)%>.   <%=st%>
       <%-- System.out.println("  SYNONYMS ARE: " + st);--%>
<%    i++;  }//1.while %><br>

       <%--CALL A FUNCTION WHICH WILL FIND IS_A TERMS --%>

<% List is_aList= new LinkedList();
is_aList=frt.FindIs_ATerms(SelOntFile,ClientKWL);
       Set isa= new HashSet(is_aList);
       Iterator Is_aLst = isa.iterator();%>
       <FONT size="3" color="#990033">RETRIEVED IS_A RELATION TERMS
ARE:</FONT>
<% int t=0;
      while (Is_aLst.hasNext()) {
          String str =(String)Is_aLst.next();%>
       <br>  <%=(t+1)%>.   <%=str%>
<%    t++;  }//1.while %>
       <br>
<%    List RelTermList= new LinkedList();
for(int j=0; j<ClientKWL.length; j++)
{
RelTermList.add(ClientKWL[j].toLowerCase());
}

List isaList= new LinkedList();
for (int f=0; f<is_aList.size(); f++){
    String isastr=is_aList.get(f).toString();
isastr=isastr.toLowerCase();
isaList.add(isastr);
}
RelTermList.addAll(isaList);


List synList= new LinkedList();
for (int r=0; r<SynonymList.size(); r++){
    String synstr=SynonymList.get(r).toString();
synstr=synstr.toLowerCase();
synList.add(synstr);}
RelTermList.addAll(synList);%>
```

```
<% IA.FindRelatedTerms frt = new IA.FindRelatedTerms();%>

<% List SynonymList= new LinkedList();
SynonymList=frt.FindSynonymTerms(SelOntFile,ClientKWL);
       Set set= new HashSet(SynonymList);
       Iterator synLst = set.iterator();%><br>
       <FONT size="3" color="#990033">RETRIEVED SYNONYM TERMS ARE:</FONT>
<% int i=0;
      while (synLst.hasNext()) {
          String st =(String)synLst.next();%>
       <br>  <%=(i+1)%>.   <%=st%>
       <%-- System.out.println("  SYNONYMS ARE: " + st);--%>
<%    i++;  }//1.while %><br>

       <%--CALL A FUNCTION WHICH WILL FIND IS_A TERMS --%>

<% List is_aList= new LinkedList();
is_aList=frt.FindIs_ATerms(SelOntFile,ClientKWL);
       Set isa= new HashSet(is_aList);
       Iterator Is_aLst = isa.iterator();%>
       <FONT size="3" color="#990033">RETRIEVED IS_A RELATION TERMS
ARE:</FONT>
<% int t=0;
      while (Is_aLst.hasNext()) {
          String str =(String)Is_aLst.next();%>
       <br>  <%=(t+1)%>.   <%=str%>
<%    t++;  }//1.while %>
       <br>
<%    List RelTermList= new LinkedList();
for(int j=0; j<ClientKWL.length; j++)
{
RelTermList.add(ClientKWL[j].toLowerCase());
}

List isaList= new LinkedList();
for (int f=0; f<is_aList.size(); f++){
   String isastr=is_aList.get(f).toString();
isastr=isastr.toLowerCase();
isaList.add(isastr);
}
RelTermList.addAll(isaList);


List synList= new LinkedList();
for (int r=0; r<SynonymList.size(); r++){
   String synstr=SynonymList.get(r).toString();
synstr=synstr.toLowerCase();
synList.add(synstr);}
RelTermList.addAll(synList);%>
```

APPENDIX E: EnterIOParameters.jsp

```
<html>
   <head>
      <title>
         ISWSF
      </title>
   </head>
   <body bgcolor="#b0c4de">
      <blockquote>
         <ul>
            <li>
            <form id="form3" runat="server">
               <hr style="width: 850px; color: #3366cc; font-style: normal; font-family: Verdana;
        height: 10px; background-color: lightsteelblue; font-variant: small-caps" align="right"
/>
               <hr style="width: 800px; height: 2px" align="right" />
               <blockquote>
                  <blockquote>
                     <p align="center"><font size="6" color="#990033">
                     INTELLIGENT SEMANTIC WEB SERVICE FINDER</font></p>
                  </blockquote>
               </blockquote>
               <hr style="width: 800px; height: 2px" align="right" />
               <hr style="width: 850px; color: #3366cc; font-style: normal; font-family: Verdana;
              height: 9px; background-color: lightsteelblue; font-variant: small-caps"
align="right" />
               <p align="center">        <b>
               <font color="#3366CC">WEB CLIENT LOOKING FOR APPROPRIATE WEB
               SERVICES </font></b></p>
            </form></li>
         </ul>
      <p align="justify"><font size="4" color="#990033">
        PLEASE ENTER INPUT/OUTPUT PARAMETERS:</font>
      </p>
          
      <fieldset style="width: 261px; height: 187px; padding: 2"><legend><p align="left"><font
color="#000080">Choose Related Terms From Here :</font></p></legend>
         <%@ page import='com.hp.hpl.jena.rdf.model.*'%>
         <%@ page import= 'com.hp.hpl.jena.ontology.OntModel'%>
         <%@ page import= 'com.hp.hpl.jena.ontology.*'%>
         <%@ page import= 'com.hp.hpl.jena.util.FileManager'%>
         <%@ page import= 'com.hp.hpl.jena.util.iterator.ExtendedIterator.*'%>
         <%@ page import ='java.io.*'%>
         <%@ page import ='java.util.List'%>
         <%@ page import ='java.util.Iterator'%>
         <%@ page import ='java.util.LinkedList'%>
         <%@ page import ='java.util.*'%>

<% String [] allterms = request.getParameterValues("AllRelatedTerms");%>
<%for(int j=0; j<allterms.length; j++){%>
         <font color="#990033"><%=(j+1)%></font>. <%=allterms[j]%><BR>
<%}%>

<% IA.IA_Match  mtch=new IA.IA_Match();

mtch.setAllTerm(allterms);%> <%
       String input= request.getParameter("inp");
            int inp = Integer.parseInt(input);
```

```
String output = request.getParameter("out");
            int oup = Integer.parseInt(output); %>

            <% mtch.setNofInp(inp);%>
            <% mtch.setNofOut(oup);%>
          <p align="left"> </p>
       </fieldset>
                 
       <fieldset style="width: 264px; height: 186px; padding: 2">
         <legend>
          <p align="center"><font color="#000080">INPUTS:</font></p>
         </legend>
          <p align="center"><font color="#000080"></font>
          <font color="#990033"> </font>
          <font size="3">
             <form method=GET action="ShowResults.jsp">
             <table>
       <%    for ( int i = 0; i < inp; i++ ) {    %>
             <TR>
                <TD> <%= i+1 %>. <input type=text size="25"
name="<%=("input"+Integer.toString(i+1))%>"></TD>
             </TR>
       <%    } %>
               </table>
          </font>
       </fieldset>      
       <fieldset style="width: 264px; height: 188px; padding: 2">
          <legend>
             <p align="center"><font color="#000080">OUTPUTS:</font></p>
          </legend>
          <p align="center"><font color="#000080"></font><font color="#990033"> </font>
          <font size="3">
          <form method=GET action="ShowResults.jsp">
          <table>
            <%    for ( int i = 0; i < oup; i++ ) {    %>
          <TR>
             <TD> <%= i+1 %>. <input type=text size="25"
name="<%=("output"+Integer.toString(i+1))%>"></TD>
          </TR>
       <%    } %>
           </table>
          </font>
       </fieldset>
        
       <p align="center">
       <input type="submit" value="Search Related Web Services" name="B1">
       </p>
       </blockquote>
     </body>
   </html>
```

APPENDIX F: ShowResults.jsp

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<html>
   <head>
      <title>
         ISWSF
      </title>
   </head>
   <body bgcolor="#b0c4de">
      <blockquote>
         <ul>
            <li>
            <form id="form2" runat="server">
               <hr style="width: 850px; color: #3366cc; font-style: normal; font-family: Verdana;
               height: 10px; background-color: lightsteelblue; font-variant: small-caps"
align="right" />
               <hr style="width: 800px; height: 2px" align="right" />
               <blockquote>
                  <blockquote>
                     <p align="center"><font size="6" color="#990033">
                     INTELLIGENT SEMANTIC WEB SERVICE FINDER</font></p>
                  </blockquote>
               </blockquote>
               <hr style="width: 800px; height: 2px" align="right" />
               <hr style="width: 850px; color: #3366cc; font-style: normal; font-family: Verdana;
               height: 9px; background-color: lightsteelblue; font-variant: small-caps"
align="right" />
               <p align="center">        <b>
               <font color="#3366CC">WEB CLIENT LOOKING FOR APPROPRIATE WEB
               SERVICES</font></b></p>
            </form>
            <%@ page import='com.hp.hpl.jena.rdf.model.*'%>
            <%@ page import= 'com.hp.hpl.jena.ontology.OntModel'%>
            <%@ page import= 'com.hp.hpl.jena.ontology.*'%>
            <%@ page import= 'com.hp.hpl.jena.util.FileManager'%>
            <%@ page import= 'com.hp.hpl.jena.util.iterator.ExtendedIterator.*'%>
            <%@ page import ='java.io.*'%>
            <%@ page import ='java.util.List'%>
            <%@ page import ='java.util.Iterator'%>
            <%@ page import ='java.util.LinkedList'%>
            <%@ page import ='java.util.*'%></li>
         </ul><fieldset style="width: 272px; height: 157px; padding: 2">
         <legend><font color="#3366CC">ALL RELATED TERMS</font></legend>

    <% IA.IA_Match  mtch=new IA.IA_Match();%>
    <% int numin=mtch.getNofInp();%>
   <%  int numout=mtch.getNofOut();%>
   <%  String[] allterm = new String[10];
      String[] inputs = new String[numin];
      String[] outputs = new String[numout];
      allterm=mtch.getAllTerm();
         for (int i = 0; i < allterm.length; i++ ) {%>
          <%= i+1 %>.<%=allterm[i]%><br>
          <%}%>
```

```
 </fieldset>          
        <fieldset style="width: 272px; height: 157px; padding: 2">
        <legend><font color="#3366CC">INPUTS</font></legend>
        <table>

<% for (int i = 0; i < numin; i++ ) {%>
        <TR>
          <TD>  <%= i+1 %>.<%
inputs[i]=request.getParameter("input"+Integer.toString(i+1));%><%=inputs[i]%></TD>
        </TR>
      <%   } %></table>
         </fieldset>       
        <fieldset style="width: 272px; height: 157px; padding: 2 ">

        <legend><font color="#3366CC">OUTPUTS</font></legend>
        <table>

<% for (int i = 0; i < numout; i++ ) {%>
        <TR>
          <TD>  <%= i+1 %>.<%
outputs[i]=request.getParameter("output"+Integer.toString(i+1));%><%=outputs[i]%></TD>
        </TR>
      <%   } %></table>
         </fieldset><br> <fieldset style="width: 950px; height: 78px; padding: 2">
          <legend><font color="#FFFFFF">RESULTS</font></legend>
          <%--MATCH RESULTS FOR INPUTS--%>
<% List inplist= new LinkedList();
inplist=mtch.runMatchForInputs(inputs);
Iterator il = inplist.iterator();
%> <FONT size="3" color="#990033">IA RETURNED MATCH SCORES FOR
INPUTS:</FONT>
<% while(il.hasNext()){
     String is =(String)il.next();%>
          <br>   <%=is%>
<%   }//1.while %> --%>
          <%--MATCH RESULTS FOR OUTPUTS--%>
<% List outlist= new LinkedList();
outlist=mtch.runMatchForOutputs(outputs);
Iterator ol = outlist.iterator();
 %>   <br> <FONT size="3" color="#990033">IA RETURNED MATCH SCORES FOR
OUTPUTS:</FONT>
<% while(ol.hasNext()){
     String os =(String)ol.next();%>
          <br>   <%=os%>
<%     }//1.while %>
        </fieldset>
      <%
List pi= new LinkedList();
List po= new LinkedList();
List pall= new LinkedList();
List urls= new LinkedList();
pi=mtch.getInpPercentage();
po=mtch.getOutPercentage();
pall=mtch.SWSPercentage();
urls=mtch.getGrdURL();%>
<% List ni= new LinkedList();
List no= new LinkedList();
ni=mtch.gInputSize();
no=mtch.gOutputSize(); %>
```

```
</ul> <fieldset style="width: 950px; height: 78px; padding: 2; background-color:#CCCCCC">
          <legend><font color="#000080">PERCENTAGES AND ACCESS POINTS OF
RETRIEVED RELATED SWSs BY IA</font></legend>
<%int k=1;
for (int p=0; p<pi.size(); p++){
       String inpp=pi.get(p).toString();
        String outp=po.get(p).toString();
        String ave=pall.get(p).toString();
        String url=urls.get(p).toString();
        String nip=ni.get(p).toString();
        String not=no.get(p).toString();
%>
          <BR> <%=k%>. SWS is:
          <br><FONT size="3" color="#990033">Inputs Similarity:</FONT>    <%=inpp%>
          <br><FONT size="3" color="#990033">Outputs Similarity:</FONT>   <%=outp%>
          <br><FONT size="3" color="#990033">Average Similarity:</FONT>   <%=ave%>
          <BR>
          THIS SWS ACCESS POINT IS : <a href="<%=url%>"><%=url%></a>
<%
float a=Float.parseFloat(inpp);
float b=Float.parseFloat(outp);
float c=Float.parseFloat(ave);
if((a==100.0) && (b==100.0) && (c==100.0)){%>
<% if((numin==Integer.parseInt(nip)) && (numout==Integer.parseInt(not))){%>
          <span style="background-color: #FFFF00"> BEST!</span>
<%}
else {%>
          <span style="background-color: #FFFF00"><br>This Isn't Enough APPROPRIATE:
NUMBER OF I/O NOT EQUAL But Approximate BEST!</span>
<%}
} else if((a==100.0) || (b==100.0)){
  if (c>50.0){%>
<% if((numin==Integer.parseInt(nip)) && (numout==Integer.parseInt(not))){%>
          <span style="background-color: #FFFF00"> FINE Enough!</span>
<%}
else {%><span style="background-color: #FFFF00"><br>Not FINE Enough!</span>
<%} } else {%> <span style="background-color: #FFFF00"><br>Not GOOD!</span><%}
}%>
<% if((a==0.0) && (b==0.0) && (c==0.0)){%>
<span style="background-color: #FFFF00"><br>WORST!</span><%}%>
          <BR>
<% ++k; } %>
           </fieldset>
       </blockquote>
      </body>
  </html>
```

APPNDIX G: StartIA.java

```
/*
 * StartIA.java
 *
 * Created on August 7, 2005, 7:07 PM
 */

package IA;
import java.awt.Container.*;
import javax.swing.*;
import com.hp.hpl.jena.rdf.model.*;
import com.hp.hpl.jena.ontology.OntModel;
import com.hp.hpl.jena.ontology.*;
import com.hp.hpl.jena.util.FileManager;
import com.hp.hpl.jena.util.iterator.ExtendedIterator;
import com.hp.hpl.jena.util.iterator.IteratorIterator.*;
import com.hp.hpl.jena.ontology.Individual;
import java.util.List;
import java.util.*;
import java.util.LinkedList;
import java.io.*;
import IA.SetClientKeyWords.*;
import java.util.Iterator;

/**
 *
 * @author duygu
 */

public class StartIA {
    private String message = "";

    public static String[] ClientKWL = new String[10];

    public static int size;
    int i=0;
    public StartIA() {
    }

    public void setMessage(String s) {
        ClientKWL[i]=s;
        ++i;
        message = s;
    }
    public String getMessage() {
        return message;
    }

    public String [] PrintClientKWL() {
        return ClientKWL;
    }

    public void setTemp(int temp) {
        size=temp;
    }
    public static void main(String args[]) {
    }
```

```
public List FindRelatedOntologyNames() {

     List OntNameList=new LinkedList();

     String [] OntFileArr= new String[4];

     OntFileArr[0] = "C:/ISWSF/Ontologies/Adhesive.owl";
     OntFileArr[1] = "C:/ISWSF/Ontologies/Financial_Investments.owl";
     OntFileArr[2] = "C:/ISWSF/Ontologies/Vehicle.owl";
     OntFileArr[3] = "C:/ISWSF/Ontologies/Software.owl";

     for (int i=0; i< OntFileArr.length; i++){

        String ontFile= OntFileArr[i];

        InputStream in = FileManager.get().open( ontFile );

        if (in == null) {
           throw new IllegalArgumentException( "File: " + ontFile + " not found");
        }

        OntModel loaded_model = ModelFactory.createOntologyModel();

        loaded_model.read(new InputStreamReader(in), "");

        Iterator it =   loaded_model.listIndividuals();

        while (it.hasNext()) {

           Individual individual = (Individual) it.next();

           String term=individual.getLocalName().toString();

           for(int j=0; j<ClientKWL.length; j++){

              if(term.equalsIgnoreCase(ClientKWL[j])) {

                 OntNameList.add(individual.getRDFType().getLocalName());

              }//if
           }//for
        }//while

     }//first for loop

     return OntNameList;
  }

}
```

APPENDIX H: SetClientKeyWords.java

```
/*
 * IntelligentAgent.java
 *
 * Created on August 1, 2005, 7:10 PM
 */
package IA;
public class SetClientKeyWords extends Object{
    private String message;
    public static String temp;

    public String getMessage() {
        return message;
    }

    public void setMessage(String s){
        temp=s;
        message = s;
    }

    public String getTemp() {
        return temp;
 }
}
```

APPENDIX I: FindRelatedTerms.java

```
/*
 * FindRelatedOntologies.java
 *
 * Created on August 6, 2005, 9:54 PM
 */
package IA;
import java.awt.Container.*;
import javax.swing.*;
import com.hp.hpl.jena.rdf.model.*;
import com.hp.hpl.jena.ontology.OntModel;
import com.hp.hpl.jena.ontology.*;
import com.hp.hpl.jena.util.FileManager;
import com.hp.hpl.jena.util.iterator.ExtendedIterator;
import com.hp.hpl.jena.util.iterator.IteratorIterator.*;
import com.hp.hpl.jena.ontology.Individual;
import java.util.List;
import java.util.*;
import java.util.LinkedList;
import java.io.*;
import IA.SetClientKeyWords.*;
import java.util.Iterator;
/**
 *
 * @author duygu
 */
public class FindRelatedTerms {
    public OntModel model;
    public FindRelatedTerms() {
        //  this.model=loadOntologies(ontFile);
    }

    public List FindSynonymTerms(String SelOntFile,String [] ClientKWL) {
        //  String SelOntFile="C:/Documents and
Settings/duygu/Desktop/Financial_Investments.owl";
        List SynonymList=new LinkedList();
        InputStream in = FileManager.get().open( SelOntFile );
        if (in == null) {
            throw new IllegalArgumentException( "File: " + SelOntFile + " not found");
        }
        OntModel model = ModelFactory.createOntologyModel();
        model.read(new InputStreamReader(in), "");
        // model.write(System.out);
        for(int i=0; i<ClientKWL.length; i++){
            Iterator it = model.listIndividuals();
            while (it.hasNext()) {
                Individual individual = (Individual) it.next();
                System.out.println("individual: " + individual.getLocalName());
                if(individual.getLocalName().equalsIgnoreCase(ClientKWL[i])){
                    Iterator propertyIt = model.listDatatypeProperties();
                    while (propertyIt.hasNext()) {
                        DatatypeProperty prop = (DatatypeProperty) propertyIt.next();
                        if (prop.getLocalName().equalsIgnoreCase("Synonym")){
                            System.out.println(" datatype property: " + prop.getLocalName());
                        Iterator valueIt = individual.listPropertyValues(prop);
                            while (valueIt.hasNext()) {
                                RDFNode value = (RDFNode) valueIt.next();
                                String str=value.toString();
```

```
        int intPos = str.indexOf("^");
                    String strTemp = str.substring(0, intPos);
                    System.out.println("    value: " + strTemp);
                    SynonymList.add(strTemp);
                }//3.while
              }//secon if
            }//2.while
          }//first if
        }//1.while
      }//first for
      return SynonymList;
    }
    public List FindIs_ATerms(String SelOntFile,String [] ClientKWL) {
      List is_aList=new LinkedList();
      InputStream in = FileManager.get().open( SelOntFile );
      if (in == null) {
        throw new IllegalArgumentException( "File: " + SelOntFile + " not found");
      }

      OntModel model = ModelFactory.createOntologyModel();
      model.read(new InputStreamReader(in), "");
      // model.write(System.out);
      for(int i=0; i<ClientKWL.length; i++){
        Iterator it = model.listIndividuals();
        while (it.hasNext()) {
          Individual individual = (Individual) it.next();
          System.out.println("individual: " + individual.getLocalName());
          if(individual.getLocalName().equalsIgnoreCase(ClientKWL[i])){
            Iterator propertyIt = model.listDatatypeProperties();
            while (propertyIt.hasNext()) {
              DatatypeProperty prop = (DatatypeProperty) propertyIt.next();
              if (prop.getLocalName().equalsIgnoreCase("Is_a")){
                System.out.println(" datatype property: " + prop.getLocalName());
                Iterator valueIt = individual.listPropertyValues(prop);
                while (valueIt.hasNext()) {
                  RDFNode value = (RDFNode) valueIt.next();
                  String str=value.toString();
                  int intPos = str.indexOf("^");
                  String strTemp = str.substring(0, intPos);
                  System.out.println("    value: " + strTemp);
                  is_aList.add(strTemp);
                }//3.while
              }//secon if
            }//2.while
          }//first if
        }//1.while
      }//first for

      return is_aList;
    }
}
```

APPENDIX J: IA_Match.java

```
/*
 * IA_Match.java
 *
 * Created on August 18, 2005, 1:37 AM
 */

package IA;
import java.awt.Container.*;
import javax.swing.*;
import com.hp.hpl.jena.rdf.model.*;
import com.hp.hpl.jena.ontology.OntModel;
import com.hp.hpl.jena.ontology.*;
import com.hp.hpl.jena.util.FileManager;
import com.hp.hpl.jena.util.iterator.ExtendedIterator;
import com.hp.hpl.jena.util.iterator.IteratorIterator.*;
import com.hp.hpl.jena.ontology.Individual;
import java.util.List;
import java.util.*;
import java.util.LinkedList;
import java.io.*;
import IA.SetClientKeyWords.*;
import java.util.Iterator;
import java.net.URI;
import org.mindswap.owl.OWLFactory;
import org.mindswap.owl.OWLIndividualList;
import org.mindswap.owl.OWLIndividual;
import org.mindswap.owl.OWLType;
import org.mindswap.owl.OWLKnowledgeBase;
import org.mindswap.owls.profile.Profile;
import org.mindswap.owls.grounding.Grounding;
import org.mindswap.owls.grounding.AtomicGrounding;
import org.mindswap.owls.grounding.WSDLAtomicGrounding;
import org.mindswap.owls.grounding.AtomicGroundingList;
import org.mindswap.owls.profile.ServiceCategory;
import org.mindswap.owls.profile.ServiceParameter;
import org.mindswap.owls.service.Service;
import org.mindswap.owls.process.ProcessList;
import org.mindswap.owls.profile.Profile;
import org.mindswap.owls.process.Process;
import org.mindswap.owls.process.InputList;
import org.mindswap.owls.process.OutputList;
import org.mindswap.owl.OWLModel;
import org.mindswap.owl.OWLClass;
import org.mindswap.owl.EntityFactory;
/**
 *
 * @author duygu
 */
public class IA_Match extends Object {
   public  String[] SWSFiles=new String[5];
   static  Profile profile;
   static  Grounding  ground;
   static Process process;
   static OWLKnowledgeBase kb;
```

```
  public static String[] allRelTerms = new String[10];
   public static int Ip=0;
   public static int Ot =0;
   public static String SelOntFile="";
   public List eachswstotoutscore=new LinkedList();
   public List eachswstotinpscore=new LinkedList();
   public void setAllTerm(String[] temp) {
      allRelTerms=temp;
   }
   public String [] getAllTerm() {
      return allRelTerms;
   }
   public void setSelOnt(String s) {
      SelOntFile=s;
   }
   public void setNofInp(int s) {
      Ip=s;
   }
   public int getNofInp() {
      return Ip;
   }
   public void setNofOut(int s) {
      Ot=s;
   }
   public int getNofOut() {
      return Ot;
   }
   public IA_Match() {
   }
   public IA_Match(String url) {
            try {
         kb= OWLFactory.createKB();
         kb.setReasoner("Pellet");
         Service service = kb.readService(url);
         profile=service.getProfile();
         process=service.getProcess();
         ground =service.getGrounding();
      } catch (Exception e) {System.err.println(e);}

   }
   public  OntModel SelOnt(){
      String  ontFile=SelOntFile;
      InputStream in = FileManager.get().open( ontFile );
      if (in == null) {
         throw new IllegalArgumentException( "File: " + ontFile + " not found");       }

      OntModel loaded_model = ModelFactory.createOntologyModel();
      loaded_model.read(new InputStreamReader(in), "");
      return loaded_model;
   }
public  List LstAllClasses(OntModel loaded_model) {
      List allClassesList=new LinkedList();
      Iterator it = loaded_model.listClasses();
//  System.out.println("ALL CLASS LIST");
      while (it.hasNext()) {
         OntClass cls=(OntClass)it.next();
String str=cls.getLocalName();
         // System.out.println("All Classes: " +  str);
```

```
allClassesList.add(str);
        }//while
        return allClassesList;
    }
    public  List getChildClass(OntModel loaded_model,String term) {
        List  getChildList=new LinkedList();
        //  System.out.println("THE "+term+" TERM ALL SUB CLASSES LIST");
        OntClass tsbcls = loaded_model.getOntClass("http://www.owl-
ontologies.com/unnamed.owl#"+term);
        for (Iterator isc = tsbcls.listSubClasses(); isc.hasNext();) {
            OntClass c = (OntClass) isc.next();
            String str=c.getLocalName();
            getChildList.add(str);
            // System.out.println(str);
        }
        return getChildList;
    }
    // Below function is listing a node(term) on ont file with its super classes
    public static List getParentClass(OntModel loaded_model,String term) {
        List getParentList=new LinkedList();
        //  System.out.println("THE "+term+" TERM ALL SUPER CLASSES LIST");
        OntClass tsbcls = loaded_model.getOntClass("http://www.owl-
ontologies.com/unnamed.owl#"+term);
        for (Iterator isc = tsbcls.listSuperClasses(); isc.hasNext();) {
            OntClass c = (OntClass) isc.next();
            String str=c.getLocalName();
            getParentList.add(str);
            //   System.out.println(str);
        }
                return getParentList;
    }
    public void SWSList(){
        String  SontFile=SelOntFile;
        //assume the UDDI is returned below SWS files
        if(SontFile.equalsIgnoreCase("C:/ISWSF/Ontologies/Financial_Investments.owl")){
            SWSFiles[0] ="C:\\ISWSF\\SWSs\\FINANCIAL\\BondPriceService.owl";
            SWSFiles[1] ="C:\\ISWSF\\SWSs\\FINANCIAL\\BondsOnlineService.owl";
            SWSFiles[2] ="C:\\ISWSF\\SWSs\\FINANCIAL\\BookInStockService.owl";
            SWSFiles[3] ="C:\\ISWSF\\SWSs\\FINANCIAL\\StockQuotesService.owl";
            SWSFiles[4] ="C:\\ISWSF\\SWSs\\FINANCIAL\\OrientedShareService.owl";        }
        if(SontFile.equalsIgnoreCase("C:/ISWSF/Ontologies/Software.owl")){
            SWSFiles[0] ="C:\\ISWSF\\SWSs\\JAVA\\JavaSyntaxService.owl";
            SWSFiles[1] ="C:\\ISWSF\\SWSs\\JAVA\\JavaCodesService.owl";
            SWSFiles[2] ="C:\\ISWSF\\SWSs\\JAVA\\JavaEducationService.owl";
            SWSFiles[3] ="C:\\ISWSF\\SWSs\\JAVA\\JavaQuestionaireService.owl";
            SWSFiles[4] ="C:\\ISWSF\\SWSs\\JAVA\\JavaInstallationService.owl";
        }
    }
    public  List runMatchForInputs(String[] ClientInp){
        SWSList();
        List eachswsinpscore=new LinkedList();
        List eachswsclientinp=new LinkedList();
        List eachswsinp=new LinkedList();
        List showinp=new LinkedList();
        float scoreinp=0;
        float eachswsinpscr=0;
        int step=1;
```

```
for(int k=0; k<SWSFiles.length; k++) {
        IA_Match o=new IA_Match(new java.io.File(SWSFiles[k]).toURI().toString());
        InputList inputList=profile.getInputs();
        showinp.add(step+".Semantic Web Service");
        step=step+1;
        for(int j=0; j<ClientInp.length; j++) {
          for(int i=0;i<inputList.size();i++) {
            String relinp=getRelation(ClientInp[j],inputList.inputAt(i).getLocalName());
            eachswsinpscr=getScore(relinp);
            scoreinp=scoreinp+eachswsinpscr;
            eachswsinpscore.add(Float.toString(eachswsinpscr));//each comparison result score
            eachswsinp.add(inputList.inputAt(i).getLocalName());//each sws input list
            eachswsclientinp.add(ClientInp[j]);//client inputs list
            showinp.add(ClientInp[j]+"-->"+inputList.inputAt(i).getLocalName()+" :
Retrieved Input Relation is= "+relinp+"   Their Score= "+eachswsinpscr+" And now total score
for  "+   SWSFiles[k]+" SWS is : "+scoreinp);
          }//3.for
        }//2.for
        eachswstotinpscore.add(Float.toString(scoreinp/ClientInp.length));//each sws total input
score divided by # of client inputs
        //bu SWS dosyanin adresini tut
        showinp.add("INPUTS SCORE WEIGHT: "+scoreinp/ClientInp.length);
        scoreinp=0;
      }//1.for

      return showinp;
  }

  public  List runMatchForOutputs(String[] ClientOut){
      SWSList();
      List eachswsoutscore=new LinkedList();
      List eachswsclientout=new LinkedList();
      List eachswsout=new LinkedList();
      List showout=new LinkedList();
      float scoreout=0;
      float eachswsoutscr=0;
      int step=1;
      for(int k=0; k<SWSFiles.length; k++) {
        IA_Match o=new IA_Match(new java.io.File(SWSFiles[k]).toURI().toString());
        OutputList outputList=profile.getOutputs();
        showout.add(step+".Semantic Web Service");
        step=step+1;
        for(int j=0; j<ClientOut.length; j++) {
          for(int i=0;i<outputList.size();i++) {
            String relout=getRelation(ClientOut[j],outputList.outputAt(i).getLocalName());
            eachswsoutscr=getScore(relout);
            scoreout=scoreout+eachswsoutscr;
            eachswsoutscore.add(Float.toString(eachswsoutscr));//each comparison result score
            eachswsout.add(outputList.outputAt(i).getLocalName());//each sws input list
            eachswsclientout.add(ClientOut[j]);//client inputs list
            showout.add(ClientOut[j]+"-->"+outputList.outputAt(i).getLocalName()+" :
Retrieved Output Relation is= "+relout+"   Their Score= "+eachswsoutscr+" And now total
score for  "+   SWSFiles[k]+" SWS is : "+scoreout);
          }//3.for
        }//2.for
eachswstotoutscore.add(Float.toString(scoreout/ClientOut.length));//each sws total input score
divided by # of client inputs
```

```
//bu SWS dosyanin adresini tut
          showout.add("OUTPUTS SCORE WEIGHT: "+scoreout/ClientOut.length);
          scoreout=0;
      }//1.for

                                    /* Iterator so = showout.iterator();
                                     while(so.hasNext()){
                                           String kw =(String)so.next();
                                           System.out.println(kw);
                                           }*/
      return showout;
   }

   public List getInpPercentage() {

      List prcInputs=new LinkedList();
      for (int p=0; p<eachswstotinpscore.size(); p++){
         String pi=eachswstotinpscore.get(p).toString();
         float pin=((Float.parseFloat(pi)*100)/3);
         prcInputs.add(Float.toString(pin));
      }
      return prcInputs;
   }

   public List getOutPercentage() {

      List prcOutputs=new LinkedList();
      for (int p=0; p<eachswstotoutscore.size(); p++){
         String po=eachswstotoutscore.get(p).toString();
         float pot=((Float.parseFloat(po)*100)/3);
         prcOutputs.add(Float.toString(pot));
      }
      return prcOutputs;
   }

   public List SWSPercentage() {

      List prcSWSs=new LinkedList();
      List prcins=new LinkedList();
      prcins= getInpPercentage();
      List prcouts=new LinkedList();
      prcouts= getOutPercentage();
      for (int p=0; p<prcins.size(); p++){
         String pi=prcins.get(p).toString();
         float pin=Float.parseFloat(pi);
         String po=prcouts.get(p).toString();
         float pot=Float.parseFloat(po);
         prcSWSs.add(Float.toString((pin+pot)/2));
      }
      return prcSWSs;
   }

   public String getRelation(String ci, String swsi) {
      ci=ci.toLowerCase();
      swsi=swsi.toLowerCase();
      String str="";
      OntModel loaded_model=SelOnt();
      List allClassesList=new LinkedList();
      List alllist= new LinkedList();
```

```
alllist=LstAllClasses(loaded_model);
      for (int f=0; f<alllist.size(); f++){
         String stra=alllist.get(f).toString();
         stra=stra.toLowerCase();
         allClassesList.add(stra); }
      if(ci.equalsIgnoreCase(swsi)) str="EXACT";
      else if((allClassesList.contains(ci)==true)&&(allClassesList.contains(swsi)==true)) {
         List getParentList=new LinkedList();
         List getPrntList=new LinkedList();
         getPrntList=getParentClass(loaded_model,ci);//look client  input parent classes
         for (int p=0; p<getPrntList.size(); p++){
            String strr=getPrntList.get(p).toString();
            strr=strr.toLowerCase();
            getParentList.add(strr); }
         List SubClassesList=new LinkedList();
         List SubClsList=new LinkedList();
         SubClsList=getChildClass(loaded_model,ci);
         for (int c=0; c<SubClsList.size(); c++){
            String strc=SubClsList.get(c).toString();
            strc=strc.toLowerCase();
            SubClassesList.add(strc);}
         if(getParentList.contains(swsi)==true) str="PLUGIN";
         else if(SubClassesList.contains(swsi)==true)  str="SUBSUME";
         else str="FAIL";
      } else str="FAIL";
      //   System.out.println(str);
      return str;
   }
      public  int getScore(String rel){
      int val=0;
      if (rel=="EXACT") val=3;
      else {
         if (rel=="PLUGIN") val=2;
         else {
            if (rel=="SUBSUME") val=1;
            else {
               if (rel=="FAIL") val=0;
            }
         }
      }
      return val;
   }
      public List getGrdURL() {
      List grdurl=new LinkedList();
      SWSList();

      for(int k=0; k<SWSFiles.length; k++) {
         IA_Match o=new IA_Match(new java.io.File(SWSFiles[k]).toURI().toString());
         List glist = ground.getAtomicGroundings();
         Iterator git = glist.iterator();
         while ( git.hasNext() ){
     WSDLAtomicGrounding atg = (WSDLAtomicGrounding)git.next();
            System.out.println("WSDLAtG URL =" + atg.getDescriptionURL() );
            //   System.out.println("WSDLAtG WSDL     =" + atg.getWSDL() );
            grdurl.add(atg.getDescriptionURL());
         }
      }
      return grdurl;
   }
```

```
  public List gInputSize() {
      SWSList();
      List SWSInpSize=new LinkedList();
      for(int k=0; k<SWSFiles.length; k++) {
         IA_Match o=new IA_Match(new java.io.File(SWSFiles[k]).toURI().toString());
         InputList inputList=profile.getInputs();
         SWSInpSize.add(Integer.toString(inputList.size()));
      }
      return SWSInpSize;
   }

   public List gOutputSize() {
      SWSList();
      List SWSOutSize=new LinkedList();
      for(int k=0; k<SWSFiles.length; k++) {
         IA_Match o=new IA_Match(new java.io.File(SWSFiles[k]).toURI().toString());
         OutputList outputList=profile.getOutputs();
         SWSOutSize.add(Integer.toString(outputList.size()));
      }
      return SWSOutSize;
   }

   public static void main(String args[]) {
   }
}
```

APPENDIX K: Finacial_Investments.owl

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
   <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
   <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
   <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
   <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]>
<rdf:RDF xmlns="http://www.owl-ontologies.com/unnamed.owl#"
    xml:base="http://www.owl-ontologies.com/unnamed.owl"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:owl="http://www.w3.org/2002/07/owl#">
   <owl:Ontology rdf:about=""/>
   <owl:Class rdf:ID="Financial_Investments"/>
   <owl:Class rdf:ID="buy">
      <rdfs:subClassOf rdf:resource="#actions"/>
   </owl:Class>
   <owl:Class rdf:ID="price">
      <rdfs:subClassOf rdf:resource="#sell"/>
   </owl:Class>
   <owl:Class rdf:ID="Price">
      <rdfs:subClassOf rdf:resource="#buy"/>
   </owl:Class>
   <owl:Class rdf:ID="sell">
      <rdfs:subClassOf rdf:resource="#actions"/>
   </owl:Class>
   <Financial_Investments rdf:ID="Goverment_Bond">
      <Is_a rdf:datatype="&xsd;string">Bond</Is_a>
   </Financial_Investments>
   <owl:Class rdf:ID="municipial">
      <rdfs:subClassOf rdf:resource="#goverment_bond"/>
   </owl:Class>
   <Financial_Investments rdf:ID="Municipial">
      <Is_a rdf:datatype="&xsd;string">Goverment_Bond</Is_a>
   </Financial_Investments>
   <owl:Class rdf:ID="instruments">
      <rdfs:subClassOf rdf:resource="#Financial_Investments"/>
   </owl:Class>
   <Financial_Investments rdf:ID="Buy">
      <Is_a rdf:datatype="&xsd;string">Process</Is_a>
      <Synonym rdf:datatype="&xsd;string">Purchase</Synonym>
   </Financial_Investments>
   <owl:Class rdf:ID="date">
      <rdfs:subClassOf rdf:resource="#Financial_Investments"/>
   </owl:Class>
<owl:Class rdf:ID="goverment_bond">
      <rdfs:subClassOf rdf:resource="#bond"/>
   </owl:Class>
   <Financial_Investments rdf:ID="Share">
      <Is_a rdf:datatype="&xsd;string">Instrument</Is_a>
      <Synonym rdf:datatype="&xsd;string">Stock</Synonym>
      <Synonym rdf:datatype="&xsd;string">Quota</Synonym>
   </Financial_Investments>
```

```xml
<owl:DatatypeProperty rdf:ID="Synonym">
    <rdfs:domain rdf:resource="#Financial_Investments"/>
    <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DatatypeProperty>
  <Financial_Investments rdf:ID="Quota">
    <Is_a rdf:datatype="&xsd;string">Instrument</Is_a>
    <Synonym rdf:datatype="&xsd;string">Stock</Synonym>
    <Synonym rdf:datatype="&xsd;string">Share</Synonym>
    <Synonym rdf:datatype="&xsd;string">Bond</Synonym>
  </Financial_Investments>
  <Financial_Investments rdf:ID="Stock">
    <Is_a rdf:datatype="&xsd;string">Instrument</Is_a>
    <Synonym rdf:datatype="&xsd;string">Bond</Synonym>
    <Synonym rdf:datatype="&xsd;string">Share</Synonym>
    <Synonym rdf:datatype="&xsd;string">Quota</Synonym>
  </Financial_Investments>
  <owl:Class rdf:ID="bond">
    <rdfs:subClassOf rdf:resource="#instruments"/>
  </owl:Class>
  <owl:Class rdf:ID="calendar_date">
    <rdfs:subClassOf rdf:resource="#date"/>
  </owl:Class>
  <Financial_Investments rdf:ID="Bond">
    <Is_a rdf:datatype="&xsd;string">Instrument</Is_a>
    <Synonym rdf:datatype="&xsd;string">Share</Synonym>
    <Synonym rdf:datatype="&xsd;string">Stock</Synonym>
  </Financial_Investments>
  <owl:Class rdf:ID="quota">
    <rdfs:subClassOf rdf:resource="#Financial_Investments"/>
  </owl:Class>
  <owl:Class rdf:ID="share">
    <rdfs:subClassOf rdf:resource="#instruments"/>
  </owl:Class>
  <Financial_Investments rdf:ID="Sell">
    <Is_a rdf:datatype="&xsd;string">Process</Is_a>
  </Financial_Investments>
  <owl:Class rdf:ID="stock">
    <rdfs:subClassOf rdf:resource="#instruments"/>
  </owl:Class>
  <owl:DatatypeProperty rdf:ID="Is_a">
    <rdfs:domain rdf:resource="#Financial_Investments"/>
    <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DatatypeProperty>
  <owl:Class rdf:ID="actions">
    <rdfs:subClassOf rdf:resource="#Financial_Investments"/>
  </owl:Class>
</rdf:RDF>
```

APPENDIX L: Adhesive.owl

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
   <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
   <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
   <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
   <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]>
<rdf:RDF xmlns="http://www.owl-ontologies.com/unnamed.owl#"
    xml:base="http://www.owl-ontologies.com/unnamed.owl"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:owl="http://www.w3.org/2002/07/owl#">
   <owl:Ontology rdf:about=""/>
   <owl:Class rdf:ID="cement">
      <rdfs:subClassOf rdf:resource="#actions"/>
   </owl:Class>
   <Adhesive rdf:ID="Stick">
      <Synonym rdf:datatype="&xsd;string">Bond</Synonym>
      <Synonym rdf:datatype="&xsd;string">Affix</Synonym>
      <Synonym rdf:datatype="&xsd;string">Bind</Synonym>
      <Synonym rdf:datatype="&xsd;string">Glue</Synonym>
      <Synonym rdf:datatype="&xsd;string">Cement</Synonym>
      <Synonym rdf:datatype="&xsd;string">Paste</Synonym>
      <Is_a rdf:datatype="&xsd;string">Adhesive</Is_a>
   </Adhesive>
   <owl:Class rdf:ID="Price">
      <rdfs:subClassOf rdf:resource="#buy"/>
   </owl:Class>
   <Adhesive rdf:ID="Glue">
      <Synonym rdf:datatype="&xsd;string">Bond</Synonym>
      <Synonym rdf:datatype="&xsd;string">Stick</Synonym>
      <Synonym rdf:datatype="&xsd;string">Affix</Synonym>
      <Synonym rdf:datatype="&xsd;string">Bind</Synonym>
      <Synonym rdf:datatype="&xsd;string">Cement</Synonym>
      <Synonym rdf:datatype="&xsd;string">Paste</Synonym>
      <Is_a rdf:datatype="&xsd;string">Adhesive</Is_a>
   </Adhesive>
   <owl:Class rdf:ID="affix">
      <rdfs:subClassOf rdf:resource="#actions"/>
   </owl:Class>
   <owl:Class rdf:ID="bond">
      <rdfs:subClassOf rdf:resource="#actions"/>
   </owl:Class>
   <owl:Class rdf:ID="Adhesive"/>
<Adhesive rdf:ID="Bind">
      <Synonym rdf:datatype="&xsd;string">Bond</Synonym>
      <Synonym rdf:datatype="&xsd;string">Stick</Synonym>
      <Synonym rdf:datatype="&xsd;string">Paste</Synonym>
      <Synonym rdf:datatype="&xsd;string">Glue</Synonym>
      <Synonym rdf:datatype="&xsd;string">Cement</Synonym>
      <Synonym rdf:datatype="&xsd;string">Affix</Synonym>
      <Is_a rdf:datatype="&xsd;string">Adhesive</Is_a>
   </Adhesive>
```

```
<Adhesive rdf:ID="Paste">
    <Synonym rdf:datatype="&xsd;string">Bond</Synonym>
    <Synonym rdf:datatype="&xsd;string">Affix</Synonym>
    <Synonym rdf:datatype="&xsd;string">Bind</Synonym>
    <Synonym rdf:datatype="&xsd;string">Glue</Synonym>
    <Synonym rdf:datatype="&xsd;string">Cement</Synonym>
    <Synonym rdf:datatype="&xsd;string">Stick</Synonym>
    <Is_a rdf:datatype="&xsd;string">Adhesive</Is_a>
  </Adhesive>
  <owl:Class rdf:ID="paste">
    <rdfs:subClassOf rdf:resource="#actions"/>
  </owl:Class>
  <owl:Class rdf:ID="actions">
    <rdfs:subClassOf rdf:resource="#Adhesive"/>
  </owl:Class>
  <owl:Class rdf:ID="price">
    <rdfs:subClassOf rdf:resource="#sell"/>
  </owl:Class>
  <owl:DatatypeProperty rdf:ID="Synonym">
    <rdfs:domain rdf:resource="#Adhesive"/>
    <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DatatypeProperty>
  <Adhesive rdf:ID="Affix">
    <Synonym rdf:datatype="&xsd;string">Bond</Synonym>
    <Synonym rdf:datatype="&xsd;string">Stick</Synonym>
    <Synonym rdf:datatype="&xsd;string">Cement</Synonym>
    <Synonym rdf:datatype="&xsd;string">Glue</Synonym>
    <Synonym rdf:datatype="&xsd;string">Paste</Synonym>
    <Synonym rdf:datatype="&xsd;string">Bind</Synonym>
    <Is_a rdf:datatype="&xsd;string">Adhesive</Is_a>
  </Adhesive>
  <Adhesive rdf:ID="adhesive">
    <Synonym rdf:datatype="&xsd;string">Cohesive</Synonym>
    <Is_a rdf:datatype="&xsd;string">Item</Is_a>
  </Adhesive>
  <owl:DatatypeProperty rdf:ID="Is_a">
    <rdfs:domain rdf:resource="#Adhesive"/>
    <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DatatypeProperty>
  <owl:Class rdf:ID="stick">
    <rdfs:subClassOf rdf:resource="#actions"/>
  </owl:Class>
  <Adhesive rdf:ID="Bond">
    <Synonym rdf:datatype="&xsd;string">Stick</Synonym>
    <Synonym rdf:datatype="&xsd;string">Affix</Synonym>
    <Synonym rdf:datatype="&xsd;string">Glue</Synonym>
    <Synonym rdf:datatype="&xsd;string">Paste</Synonym>
    <Synonym rdf:datatype="&xsd;string">Cement</Synonym>
    <Synonym rdf:datatype="&xsd;string">Bind</Synonym>
    <Is_a rdf:datatype="&xsd;string">Adhesive</Is_a>
  </Adhesive>
  <owl:Class rdf:ID="buy">
    <rdfs:subClassOf rdf:resource="#actions"/>
  </owl:Class>
  <owl:Class rdf:ID="glue">
    <rdfs:subClassOf rdf:resource="#actions"/>
  </owl:Class>
```

```
<owl:Class rdf:ID="sell">
      <rdfs:subClassOf rdf:resource="#actions"/>
   </owl:Class>
   <owl:Class rdf:ID="bind">
      <rdfs:subClassOf rdf:resource="#actions"/>
   </owl:Class>
   <Adhesive rdf:ID="Cement">
      <Synonym rdf:datatype="&xsd;string">Bond</Synonym>
      <Synonym rdf:datatype="&xsd;string">Bind</Synonym>
      <Synonym rdf:datatype="&xsd;string">Affix</Synonym>
      <Synonym rdf:datatype="&xsd;string">Glue</Synonym>
      <Synonym rdf:datatype="&xsd;string">Paste</Synonym>
      <Synonym rdf:datatype="&xsd;string">Stick</Synonym>
      <Is_a rdf:datatype="&xsd;string">Adhesive</Is_a>
   </Adhesive>
</rdf:RDF>
```

APPENDIX M: Vehicle.owl

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE rdf:RDF [
   <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
   <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
   <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
   <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >]
>
<rdf:RDF xmlns="http://www.owl-ontologies.com/unnamed.owl#"
    xml:base="http://www.owl-ontologies.com/unnamed.owl"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:owl="http://www.w3.org/2002/07/owl#">
   <owl:Ontology rdf:about=""/>

   <owl:Class rdf:ID="train">
      <rdfs:subClassOf rdf:resource="#Vehicle"/>
   </owl:Class>

   <owl:Class rdf:ID="suv">
      <rdfs:subClassOf rdf:resource="#car"/>
   </owl:Class>

   <Vehicle rdf:ID="Ship">
      <Synonym rdf:datatype="&xsd;string">Boat</Synonym>
      <Synonym rdf:datatype="&xsd;string">Vessel</Synonym>
      <Is_a rdf:datatype="&xsd;string">Vehicle</Is_a>
   </Vehicle>

   <owl:DatatypeProperty rdf:ID="Synonym">
      <rdfs:domain rdf:resource="#Vehicle"/>
   </owl:DatatypeProperty>

   <Vehicle rdf:ID="Train">
      <Is_a rdf:datatype="&xsd;string">Vehicle</Is_a>
   </Vehicle>

   <owl:Class rdf:ID="car">
      <rdfs:subClassOf rdf:resource="#Vehicle"/>
   </owl:Class>

   <Vehicle rdf:ID="Truck">
      <Synonym rdf:datatype="&xsd;string">Lorry</Synonym>
      <Is_a rdf:datatype="&xsd;string">Vehicle</Is_a>
   </Vehicle>

   <Vehicle rdf:ID="Boat">
      <Synonym rdf:datatype="&xsd;string">Craft</Synonym>
      <Synonym rdf:datatype="&xsd;string">Ship</Synonym>
      <Is_a rdf:datatype="&xsd;string">Vehicle</Is_a>
   </Vehicle>

   <owl:Class rdf:ID="boat">
      <rdfs:subClassOf rdf:resource="#ship"/>
```

```
</owl:Class>

  <owl:DatatypeProperty rdf:ID="Is_a">
    <rdfs:domain rdf:resource="#Vehicle"/>
  </owl:DatatypeProperty>

  <Vehicle rdf:ID="Car">
    <Synonym rdf:datatype="&xsd;string">Automobile</Synonym>
    <Is_a rdf:datatype="&xsd;string">Vehicle</Is_a>
  </Vehicle>

  <Vehicle rdf:ID="Station_Wagon">
    <Is_a rdf:datatype="&xsd;string">Car</Is_a>
  </Vehicle>

  <owl:Class rdf:ID="station_wagon">
    <rdfs:subClassOf rdf:resource="#car"/>
  </owl:Class>

  <owl:Class rdf:ID="ship">
    <rdfs:subClassOf rdf:resource="#Vehicle"/>
  </owl:Class>

  <Vehicle rdf:ID="vehicle">
    <Synonym rdf:datatype="&xsd;string">Conveyance</Synonym>
  </Vehicle>

  <Vehicle rdf:ID="Bus">
    <Synonym rdf:datatype="&xsd;string">Coach</Synonym>
    <Is_a rdf:datatype="&xsd;string">Vehicle</Is_a>
  </Vehicle>

  <owl:Class rdf:ID="Vehicle"/>

  <owl:Class rdf:ID="bus">
    <rdfs:subClassOf rdf:resource="#Vehicle"/>
  </owl:Class>

  <owl:Class rdf:ID="truck">
    <rdfs:subClassOf rdf:resource="#Vehicle"/>
  </owl:Class>

  <Vehicle rdf:ID="Sedan">
    <Is_a rdf:datatype="&xsd;string">Car</Is_a>
  </Vehicle>

  <owl:Class rdf:ID="sedan">
    <rdfs:subClassOf rdf:resource="#car"/>
  </owl:Class>

<Vehicle rdf:ID="SUV">
<Is_a rdf:datatype="&xsd;string">Car</Is_a>
</Vehicle>

</rdf:RDF>
```

APPENDIX N: Software.owl

```
<?xml version="1.0"?>

<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
   xmlns="http://www.owl-ontologies.com/unnamed.owl#"
 xml:base="http://www.owl-ontologies.com/unnamed.owl">
 <owl:Ontology rdf:about=""/>

 <owl:Class rdf:ID="tags">
  <rdfs:subClassOf>
   <owl:Class rdf:ID="syntax"/>
  </rdfs:subClassOf>
 </owl:Class>

 <owl:Class rdf:ID="Software"/>

 <owl:Class rdf:about="#syntax">
  <rdfs:subClassOf>
   <owl:Class rdf:ID="java"/>
  </rdfs:subClassOf>
 </owl:Class>

 <owl:Class rdf:ID="arrays">
  <rdfs:subClassOf rdf:resource="#syntax"/>
 </owl:Class>

 <owl:Class rdf:ID="java.net">
  <rdfs:subClassOf>
   <owl:Class rdf:about="#java"/>
  </rdfs:subClassOf>
 </owl:Class>

 <owl:Class rdf:ID="parameters">
  <rdfs:subClassOf rdf:resource="#syntax"/>
 </owl:Class>
 <owl:Class rdf:ID="tutorials">
  <rdfs:subClassOf rdf:resource="#Software"/>
 </owl:Class>

 <owl:Class rdf:about="#java">
  <rdfs:subClassOf>
   <owl:Class rdf:ID="tool"/>
  </rdfs:subClassOf>
 </owl:Class>

 <owl:Class rdf:about="#tool">
  <rdfs:subClassOf rdf:resource="#Software"/>
 </owl:Class>

 <owl:Class rdf:ID="MS.NET">
  <rdfs:subClassOf rdf:resource="#tool"/>
 </owl:Class>
```

```
<owl:Class rdf:ID="c_tutorial">
    <rdfs:subClassOf rdf:resource="#tutorials"/>
  </owl:Class>

  <owl:Class rdf:ID="JSP">
    <rdfs:subClassOf rdf:resource="#java"/>
  </owl:Class>

  <owl:Class rdf:ID="netbeans4.0">
    <rdfs:subClassOf rdf:resource="#java"/>
  </owl:Class>

  <owl:Class rdf:ID="programs">
    <rdfs:subClassOf rdf:resource="#java"/>
  </owl:Class>

  <owl:Class rdf:ID="examples">
    <rdfs:subClassOf rdf:resource="#java"/>
  </owl:Class>

  <owl:Class rdf:ID="c_sharp.net">
    <rdfs:subClassOf rdf:resource="#MS.NET"/>
  </owl:Class>

  <owl:Class rdf:ID="classes">
    <rdfs:subClassOf rdf:resource="#syntax"/>
  </owl:Class>

  <owl:Class rdf:ID="installation">
    <rdfs:subClassOf rdf:resource="#java"/>
  </owl:Class>

  <owl:Class rdf:ID="vb.net">
    <rdfs:subClassOf rdf:resource="#MS.NET"/>
  </owl:Class>

  <owl:Class rdf:ID="java_tutorial">
    <rdfs:subClassOf rdf:resource="#tutorials"/>
  </owl:Class>

  <owl:Class rdf:ID="netbeans4.1">
    <rdfs:subClassOf rdf:resource="#java"/>
  </owl:Class>

  <owl:DatatypeProperty rdf:ID="Is_a">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#Software"/>
  </owl:DatatypeProperty>

  <owl:DatatypeProperty rdf:ID="Synonym">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#Software"/>
  </owl:DatatypeProperty>

  <Software rdf:ID="C_Sharp">
    <Is_a rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Tool</Is_a>
```

```
<Is_a rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Computer Language</Is_a>
  <Is_a rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Software</Is_a>
</Software>
<Software rdf:ID="software">
  <Is_a rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Program</Is_a>
</Software>
<Software rdf:ID="VB.NET">
  <Is_a rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Computer Language</Is_a>
  <Is_a rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Software</Is_a>
  <Is_a rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Tool</Is_a>
</Software>
<Software rdf:ID="Tool">
  <Is_a rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >VB.NET</Is_a>
  <Is_a rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Java</Is_a>
  <Synonym rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Computer Tool</Synonym>
  <Is_a rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >C</Is_a>
</Software>
<Software rdf:ID="C">
  <Is_a rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Tool</Is_a>
  <Is_a rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Computer Language</Is_a>
  <Is_a rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Software</Is_a>
</Software>

<Software rdf:ID="Tutorial">
  <Is_a rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Conference</Is_a>
  <Synonym rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Lecture</Synonym>
</Software>

<Software rdf:ID="Java">
  <Is_a rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Software</Is_a>
  <Is_a rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Computer Language</Is_a>
  <Is_a rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Tool</Is_a>
</Software>
</rdf:RDF>
```